Visual Interaction Techniques for Courseware Production and Presentation

by

Lam Shing Yung, Anton

Supervised by
Dr. C. S. Chang

Department of Computer Science
The Chinese University of Hong Kong
May, 1991

A Thesis submitted in partial fulfillment of the requirements
for the Degree of Master of Philosophy in
the Chinese University of Hong Kong

Acknowledgements

I would like to thank Dr. C. S. Chang, my supervisor, for his support and ideas. Moreover, I am honoured to dedicate this thesis to my parents, to whom I am deeply indebted for their encouragement.

Abstract

This thesis describes the project of design and implementation of a Courseware Production and Presentation System, and the research and situations that led to its development. The system is implemented on OS/2 Presentation Manager, which is a graphical, windowed environment.

The presentation materials are viewed as objects of different classes. Novice users of the system can use predefined objects and operations that act on those objects to prepare ordinary presentations. Experienced users, which found the objects and operations provided inadequate, can define objects and operations themselves.

An Active-Object-Set Model is introduced in order to simplify the relations between objects and operations. Based on this model, interaction techniques are devised so that users can define new object classes, which are based on primitive objects, and new operations, which are based on primitive operations, through direct manipulations, menu selections, dialogue box interactions, etc.

The system demonstrates that it is possible to define the run-time behaviour of visual objects without any textual programming. The process involves interactions that are familiar to a user of window-based applications.

The possibility to extend the model to other application areas is also discussed. Two of the areas are Visual-Object Oriented Systems and User Interface Management Systems.

Table of Contents

Chapter 1
Introduction

The most fundamental objective of this research is to develop a Courseware Production and Presentation System. In order to satisfy requirements such as user-friendliness and extensibility, the software is implemented on a windowing system supporting popular interaction techniques. Due to the complexity of the windowing environment, an Active-Set-Model is proposed as a guideline to help the design and implementation of windowing applications in which visual objects manipulation is the primary function.

The Active-Set-Model simplifies the interactions between different visual objects. The physical and run-time properties of an object is packed into one single unit. Not only programmer benefits from this model, by applying appropriate visual interaction techniques, users can define and override object properties, thus extending the functionality.

The model is currently applied to the Courseware Production and Presentation System, but it can be extended to other visual-object oriented applications. One important example is the User Interface Management System (UIMS), which is gaining much attention recently. Moreover, the use of visual expressions to define the run-time behavior of objects is one important category of visual programming [36], although the current techniques can only be applied to very high level and general manipulative actions.

1.1. Motivations for Presentation System

Presentation is the most common and effective way to deliver information and ideas to the audience in a formalized and organized manner. The quality of the presentation depends on how the materials are prepared and organized, and how these materials are presented. These directly affect the perception of the audience.

A high quality presentation, which carefully makes use of text, diagrams, pictures and other media, usually needs a longer period to prepare. For small scale presentations, or frequent presentations with minor changes, it is not worth paying so much time and effort to create or modify the presentation. In fact, many authors sacrifice quality for time saving. For examples, the authors use handwritten text and hand-drawn diagrams in single medium to avoid formatting and printing, and to avoid integrating pictures, photographs and sound.

As computers are being more popular, together with the widely used word-processors, people realize that computers can be used to produce and maintain presentation materials rather easily and quickly. Moreover, with the evolution of *What-You-See-Is-What-You-Get* (WYSIWYG) packages, the display on a monitor can be used for the presentation directly, by projecting the materials on to a large screen, thus, the need for softwares to support presentations becomes imminent.

1.2. Shortcomings of Traditional Method

Slides and transparencies are usually used in presentations even up to now when computers are widely used. This kind of traditional method is static, and usually black and white, though it was once the most effective way, when compared with the even older method of using blackboards. When more dynamic presentations are needed, the only way was to produce a movie or video, which is non-interactive and very expensive.

The traditional method of presentations suffered from the following major disadvantages:

1. The production process.

   To produce slides or transparencies (except hand-written ones), the materials are to be prepared on separate sheets first. Then they are to be photoed (for slides) or photocopied (for transparencies), and processed before they can be actually used. The time taken is usually long and the materials cannot be easily modified.

2. Reuse of materials.

   It is difficult to extract and reuse part of the existing materials which exist on slides or transparencies. The only way to reuse them is to make another copy and integrate them with new materials through cut and paste.

3. Integration of different media.

   Materials of different media are usually produced with different methods: word-processor for textual materials, drawing program for diagrams, scanning program for bitmaps and pictures. The integration of text and diagrams onto a single sheet is usually done by cut and paste. The layout of materials cannot be modified easily. It is especially inconvenient for textual materials for which reformatting to fill different paragraph widths is desirable.

4. Use of color.

   It is not difficult though expensive to prepare color slides, provided that the original materials are in color. The processing of color transparencies is even more expensive, which is not affordable for schools and universities.

5. Quality.

   Slide is the best medium for photos and color pictures because of its high resolution. But the final output quality is usually determined by the photographer's skill. Photos and pictures are needed to be photocopied first before they can be put on to transparencies. The quality is usually not so good.

6. Dynamic presentation style.

   Presentation using slides and transparencies are performed in strict sequence, with little flexibility.

7. Cost.

   The cost of producing slides and transparencies mainly depends on the number of slides or transparencies to be processed. Except for color transparencies, the cost is rather low.

1.3. Computerized Courseware Production and Presentation System

Although the main functions involved in the courseware production process (i.e. material gathering, organizing and layout) require human intelligence (because only the author knows what he/she wants to present and how to present), the computer should take a vital part in all the three sub-tasks. Electronic libraries or Hypertext databases [21] would provide very fast routes to search, view and extract raw materials. Word-processors with outline editing facility, or hypertext authoring systems could certainly help in organizing the materials into different sections or chapters. Graphics editors are useful in the layout process. The production facility provided by our system serve as an example editor for the layout process.

Most of the presentation media have strong hardware requirements, either in the production phase or the presentation phase; and some are laborious. With the decrease in price of computer hardware and increase in popularity of the usage, the computer should provide an integrated solution to the production and presentation process.

Using computer as the presentation tool, one can enjoy the following benefits:

1. Creation of Material

   The creation of textual and graphical materials has been widely used in word-processing, drawing and desktop publishing programs.

2. Integration of Text and Graphics

   Existing diagrams, pictures, photographs, and text can be integrated into the same presentation rather easily, in a WYSIWYG manner.

3. Multimedia

   Video and digitized audio can be integrated into the presentation also.

4. Color

   If a color monitor is available, colorful presentation is possible with software support.

5. Animation (Dynamic Presentation)

   This feature can never be found in static presentation media like slides. To make smooth animation, a fast CPU is desired.

6. Easy Maintenance

   Existing material can be reused and modification is easy, without the time-consuming cut-and-paste process.

7. Low Cost

   The fixed cost depends on the computer hardware cost, which is decreasing continuously, and the presentation software cost. Moreover, a computer is multi-purpose. The operating cost involves the labour and hardware maintenance cost. It is extremely cheap when compared to the production cost of slides and transparencies.

8. Special Transition Effect

   The special transition effects which are common only in movies, such as fade in/out, can be done on a computer display while it is not possible in traditional presentation.

9. Interactivity

   The author can interact with the system during a presentation: interactive feedback (e.g. highlight a selected block when mouse is clicked over it) and alternative routes that depend on run-time status (e.g. click on different buttons to go to different locations in the courseware).

1.4. Hardware Advances

At the very beginning, a presentation system simply resembles a slide projector. The hardware requirement is little, only a display unit supporting raster graphics display is needed. As more sophisticated systems emerged, more advanced hardware are needed.

Generally, the following hardware are required for a sophisticated presentation system:

1. A large hard disk with short access time for the storage of multimedia information such as bitmaps and digitized sound data. An estimate of twenty megabytes are needed to store sixty-four full-screen 8-bit colored bitmaps.

2. A fast computer, capable of running a sophisticated windowing system in graphics mode, for the smooth interaction with the end-user. Generally, a 386-based IBM personal computer or its compatibles, or a Macintosh SE are capable of doing so.

3. A high-resolution color monitor for the display of pictures, the scanned images, and even video. A 8-bit per pixel, 70 pixels per inch color monitor is acceptable.

4. A scanner for the digitization of printed images.

5. A digital camera[1] for the digitization of three dimensional objects such as, landscapes and human beings.

The above configuration was rare in early microcomputer systems, even at the minicomputer platform, eight mega-bytes of main memory seemed to be adequate at that time. Today, plugging four mega-bytes into a microcomputer seems to be usual. Some have sixteen, in fact. Together with a huge computing power microprocessor and a high-resolution graphics display, the powerful graphics capabilities required by a presentation system is available. One can realize that most of the hardware listed above are very suitable for the creation, manipulation and storing of multimedia information which involves text, pictures, photographs and digitized audio signal.

Currently, a high-speed laptop/notebook computer with large secondary storage (e.g. 80Mb, 100Mb) together with a color LCD display panel (up to 640 x

1 One example is the Canon RC-250 Still Video Camera.

480 pixels) make the presentation system portable. And high-end RGB projectors are suitable for setting up permanent presentation sites. Moreover, removable hard disks can let different users share the same hardware while they can take away their data in a single pack.

Laptop or notebook computers that are faster, smaller and lighter than personal computers are going to the market everyday. Obviously, the hardware advances make ideal presentation systems possible, especially those running on a windowing system, which supports various interaction techniques.

1.5. Windowed, Graphical Applications

One major requirement of the Courseware Production and Presentation System is to provide a user-friendly environment for the user. As the Graphical User Interface (GUI) is gaining much more recognitions for its consistent and easy-to-use interface, the OS/2 Presentation Manager GUI is selected as the development platform.

User interfaces are switching from the single command line environment towards a windowed, graphical environment. This trend can be realized on the Workstations platform, such as DECWindow, SPARC's OpenWindow and NeXT's NextStep, and on the micro-computer platform, such as OS/2 Presentation Manager, Microsoft Windows and Macintosh.

Although there are many Graphical User Interfaces, they are all evolved from the prototypes developed at the Xerox Palo Alto Research Centre in the 70s. They all share similar interaction objects such as scroll bars, menu bars, buttons, etc. Most interactions with the application/system are done through direct manipulations, which resemble the real-world situations closely. Users are able to pick up the control in a relatively short period, and the users' need to memorize the commands is greatly reduced. The consistent user interface among different applications, and even among different systems greatly shorten the time needed to learn a new application. Users are able to transfer their knowledge and skills from one application/system to another without much external help.

In the GUI environment, the learning curve for a novice user to accustom to the system is flattened. But it is not the case for the programmers who develop the applications for such an environment.

Event-driven programming is commonly used in a windowing environment. Program developers no longer have active control over the flow of the applications, rather, they need to tackle different relevant events happening to the application by providing call-back procedures. The user-friendly interface transfers the burden of controlling the machine smoothly to program developers. User Interface Management Systems are being developed to help the creation and management of all aspects of user interfaces. But other than the general user interface, issues such as providing dynamic feedback are still hard to program. The same problem are faced in the development of the Courseware Production and Presentation System. The development of the Active-Set-Model is an attempt to provide a simpler programming model for windowed, graphical applications.

1.6. Interaction Techniques

An interaction technique is to input commands, values, names, etc. by using certain type of input device, such as mouse, keyboard, tablet, etc. In a windowing system, the various physical input devices are usually abstracted to locator devices, keyboard devices, valuator devices, etc. These logical devices are realized in the user interfaces by menus, buttons, list boxes, scrollbars, virtual keyboards, etc.

The commonly used user interface objects have the following features:

1. Resemblance of real world objects.
   Radio button is an excellent example of user interface objects that resembles closely to the real world equivalent. When a user depresses one button, the others reset. With the help of two and a half dimension graphics, user can 'really' see the state changes of the buttons, thus made the concept easily be accepted (refer to figure 15).
   Some interaction objects do not really have their real-world equivalent, but users can easily get the concept from the name of the interaction object and a few trials. One example is the pull-down menu, which from its name, users can easily guess that one item from a list of choices can be selected.

2. Provision of dynamic feedback
   Dynamic feedback is a technique to show the progress or state of a certain interaction. For example, the scroll box inside a scroll bar shows the relative position of the viewport within the whole document, and it will change dynamically during user manipulation. Other examples are: highlight bar during menu selection, change of button shapes when depressed, change of pointer shape during mouse movement, etc.

Generally, an interaction technique refers to a single interaction with an user interface object. In our research, however, interaction techniques refer to the boardest definition, that is, through the use of physical devices, a certain type of value or command is input, though the interaction involved may not be limited to one single step, but may consist of a series of traditional interaction techniques.

1.7. Research Objectives

The two objectives for this project are:

1. To develop a user friendly courseware production environment.
   The environment for production of courseware will allow users to integrate different media in a convenient and consistent way. The integrated materials can be further edited. The system will act as a prototype demonstrating the capabilities of such an environment. Text, basic geometric lines and shapes, and bitmaps are supported, but not sound and video for the time being.

2. To devise interaction techniques for the easy specifications of new manipulative operations.
   In the presentation part of the system, the user will be allowed to define new manipulative operations on the materials, thus to extend the functions of the system. The definition process should be as simple as possible. The techniques involved should be those used in manipulating the text or graphics objects in the production part, that is, those the users are already familiar with. Direct manipulation techniques will be used whenever appropriate.

Chapter 2
Existing Products and Related Research

Using computer to produce presentation materials is not a new idea. Word-processors combined with laser printers with various high-resolution typefaces and sizes are used to typeset textual materials. Presentation graphics softwares help to visualize information through charts. Diagrams and illustrations can be produced with different available software. But using computers directly as the presentation medium is rare. Lack of software support is the main obstacle.

In this chapter, several available softwares with presentation capabilities will be discussed. We will focus on the integration of materials, types of media supported, presentation styles supported and the interaction techniques involved.

Apart from the applications on the market, we will discuss the academic researches that are related to the issues or problems faced in our project. These researches are Authoring Systems, Visual Programming and User Interface Management Systems.

2.1. Existing Products

Six existing products that have the features of a Courseware Production and Presentation System are evaluated in this section. All these presentation packages are run on personal computers. Nowadays, laptops or notebook personal computers are becoming ever increasingly popular.

2.1.1. PRESENT Slide Presentation System

The PRESENT system is an example of early primitive presentation system. It closely resembles the work of a slide projector. First, the author should prepare all the images (prepare slides). Then, the images are arranged in the desired order (put slides on a tray). Finally, they are presented one by one (projected to screen). Simple screen transition effects are supported (e.g. fade-in, fade-out).

This system does not support any integration of materials. It only captures the screen image, which is generated by other software, and save it as a bitmap. So bitmap is the only media it supports, although there may be text and diagrams within, it all depends on the software that produced the image. Maintaining the presentation, that is, to modify the material, is quite troublesome because several different softwares are involved.

The software is highly device dependent because it will access the display hardware directly to get the screen image. And it may be incompatible with the software that generates the desired image. Because it is compatible with the Enhanced Graphics Adapter (EGA) standard, color 'slides' can be produced.

Because of the simplicity of the software, the users can interact with it very easily. All functions are chosen from various menus. The number of special keys to be remembered are little, but mouse interaction is not supported.

2.1.2. Harvard Graphics

This software is used for the creation of business presentation graphics. Different charts (line graph, bar chart, pie chart, etc) can be created, and simple text screen can be edited. The presentation part is just one of the various functions available. Like the PRESENT system, a slide show can be created showing charts and text screens one by one. Color slides are supported.

The system does not support the integration of arbitrary text and graphics. The type of materials the system can produce are charts and text screens only. Still, it can satisfy the basic presentation needs in a business environment. The interaction techniques involved are simple menu choices.

2.1.3. HyperCard

HyperCard is actually a mini-hypertext system rather than a presentation system. Cards are organized into stacks and linkages can be established between cards on the same or different stacks. HyperCard implements many of the hypertext concepts such as linking one card to another, providing buttons for invoking different actions, etc. by showing cards on a stack continuously, a slide show is resembled.

Acting as a presentation system, it is more powerful than the last two softwares that it has a WYSIWYG editing environment for the creation and integration of graphics and text. Materials prepared by other applications can be obtained via the cut-and-paste operation. Moreover, it can integrate sound and video.

The presentation produced can be highly interactive. If buttons, which link to other cards, are used appropriately, the users not only can control the timing, but also manipulate the presentation sequence as they wish. They can go back and forth, or jump to another related cards by clicking on pre-defined buttons or selecting from menus.

The most significant shortcoming of using HyperCard is that it supports only black and white materials, thus cannot utilize the high-resolution color display to achieve a more impressive presentation. Moreover, the size of each card is fixed to fit in a small screen used by the smaller Macintosh model. To specify the linkages between various cards is quite difficult, even from a programmer's point of view. One can easily get loss when jumping from cards to cards. The problem of disorientation may be solved by a web-like overview.

2.1.4. Macromind Director

Macromind Director which runs on Macintosh computers is a sophisticated software for producing presentations. It treats every piece of presentation material as an object, and each object, in turn, is a cast member. The author can put different cast members on to the stage and specify where each cast member should go at different times. Each time-frame within a presentation is to be specified.

Text, graphics, sound and video are all supported. Like Hypercard, a highly interactive, direct manipulation interface is provided for the creation and editing of text and geometric graphics objects. And materials from other applications can be imported. In the latest version, three dimensional objects can also be integrated. The system also takes full advantages of the color display.

The application treats each piece of material as a cast member which performs some actions during its existence on the stage, so users can specify the movement of a cast member, that is, the application supports simple animations. The animation of a cast member can be specified by one of the two methods: real-time recording and space-to-time specification. Real-time recording traces the movement of objects manipulated by the author. Space-to-time method allows the author to specify the position of an object at each time-frame. A score window records all the cast members' actions at any time. The job of the author is similar to the job of a film director.

One shortcoming of Macromind Director is that it can only produce sequential and non-interactive presentations. The author can preset wait points in the presentation but cannot pause and resume the presentation at arbitrary points. It is still very suitable for well planned and self-running presentations, which need no human intervention. Another shortcoming is the complexity of the system, making novice users unwilling to learn the features.

2.1.5. Authorware Professional

Authorware Professional is an authoring tool for users to design and create interactive courseware through the use of a set of user-friendly tools. It allows users to integrate text, graphics, animations and even video. Like other Macintosh software, users can enter text, draw lines and rectangles, change their attributes in a WYSIWYG manner. Graphics files can be imported to the application as another object.

The presentation of the materials are specified by a flow-line of icons. Each type of icons represents a predefined action. One or more objects are associated with an icon. For example, the display icon shows the associated objects on the screen; the animation icon moves the associated objects along a certain path; and the erase icon clear the associated objects using a predefined style (e.g. fade-out, zoom to line).

The fundamental usage of the system is to produce courseware, which is highly interactive. A user can have full control over the timing of the presentation, and enable branching through the 'interaction icon'. When the presentation comes to a point where an interaction icon is inserted, the system will wait for the user's selection, based on the selection, the system will continue the presentation at different branches.

Users of Authorware Professional need to place the presentation sequence in a higher priority than the presentation materials, i.e. users must specify the presentation sequence before they fill in the contents. It is hard to modify the presentation sequence (especially breaking one frame into smaller sequence) after it has been defined. This approach is not natural because it reverses the importance of the actual content and the presentation styles used.

Other than courseware production and presentation, the system supports a variety of drills and interactive exercises. But this is out of the scope of this project.

2.1.6. PageMaker, Ventura and MacDraw

Although PageMaker Ventura and MacDraw are not dynamic presentation softwares, but they have features of the production part of the project, so some discussions will be made.

PageMaker and Ventura are popular page layout programs, which facilitate the integration of text, graphics, diagrams, etc. on printed pages. They are, on one hand, word-processors, which users can create paragraphs of text, change their typefaces, sizes and styles, align them to margins, etc.; on the other hand, the programs provide much more powerful functions for the arrangement of different objects on pages, such as putting a scanned photograph on arbitrary position within a page, then allow text to flow around the photograph in a certain manner.

MacDraw is an application for the production of general technical drawing. It allows the creation of various geometric lines and shapes, optionally filled with patterns.

All these applications operate in a WYSIWYG manner, the typefaces, sizes and styles of the text, the geometric shapes, the pictures, the photographs, etc. can be viewed on the screen, and can be manipulated directly using the mouse.

The final products of these applications are hardcopies of the material, so the main concern of the applications are the static attributes of the materials, i.e. the appearance. Moreover, since color printers are not so popular, these programs generally do not support color. What we concern most about these kind of programs are the interaction techniques employed in the integration of different kind of materials. In fact, the interaction techniques used in the production part of our system are very similar to those used in these programs. One can view it as a simplified version which targets for the screen, not for pieces of paper.

2.1.7. Summary

In summary, there are well developed interaction techniques for the creation, integration and manipulation (e.g. resize, edit, change color) of materials of different media (excluding sound and video, which are still hard to be manipulated by using a mouse). But these techniques are related only to the static part of the objects. The interaction techniques used in specifying the attributes of the dynamic part of an object are still very limited.

2.2. Related Research

The design of our system is influenced by researches in the following areas: Authoring Systems, User Interface Management System and Visual Programming.

2.2.1. Authoring Systems

Authoring systems aim at providing a set of easy-to-use tools for designing and creating interactive software applications. They gain particular interest in the field of Computer Aided Instruction (CAI). Authoring systems of this kind allow users to create courseware, control the presentation sequence, accept students' response, analyze students' performance and manage student records. A common objective found in these systems is to free authors from the burden of programming.

Kearsley [22] characterized an authoring system with the following four levels:

1. Content Creation

   This innermost level refers to the input, arrangement and modification of different media used in the courseware. In advanced systems, WYSIWYG editors, which supports graphics editing, are used.

2. Lesson Definition

   This level concerns with the specification of the structure of the lessons, which involves the presentation sequence and the styles used, and how students would interact with the system.

3. Course Management

   In this level, users can select a particular instructional strategy, specify the response data to be collected, document the courseware, etc.

4. Authoring Environment

   This level defines the nature of the interactions between the authoring system program and the author. The environment should be different for users of different skill level, or different instructional purposes.

The first characteristic is obviously what we are concerned most, while the others will not be investigated.

Although interactive graphics are effective in creating the courseware content, the techniques involved are usually borrowed from the research results of computer graphics. What we are interested, therefore, in the field of authoring systems is the presentation model, that is, how coursewares are presented and how the users interact with them.

Frame-based approach is commonly used [1] [14]. The materials are organized and presented within a frame, the size of which is limited by the screen size. As windowing environment becomes popular, the use of overlapping windows, which can be viewed as other frames, facilitates the presentation in the way that one can view more information at the same time, with little confusion.

Interactivity is another important issue in courseware design [26] [3]. Authoring systems always try to provide a better interface between students and the system. A lot of interaction paradigms are provided, such as, response options like text, numeric and touch/click actions, feedback and branching. Students will not just sit and look, they will participate and even alter the instruction sequence by providing different answers to questions, or by choosing different routes predefined by the author. Another field of research that interactivity plays a vital part is the area of Hypertext.

Interactivity is an important theme in our research. The interactions between authors (teachers) and the system is emphasized, that is, we aimed at providing a highly interactive and intuitive environment for the teachers to create and present the courseware. There will be no facility provided for getting the students' responses.

Most authoring systems require the use of authoring languages, which are textual. Users need to learn some concepts of programming before they can use the languages. The language approach is certainly not applicable to a system which expected to be widely used by non-computer professionals.

2.2.2. User Interface Management System (UIMS)

User Interface Management System becomes a hot topic as the windowed, graphical user interface is gaining more acceptance, while more and more windowing applications developers find difficulties in creating, maintaining and manipulating the user interface. A comprehensive UIMS is a tool that helps an application developer create and manage all aspects of user interfaces [32].

A comprehensive UIMS consists of three major components:

1. Interaction Technique Library
   This kind of library is generally provided by the windowing system. Examples are the Software Development Kit of Microsoft Windows, OS/2 Toolkit, XToolkit of X Windows and DecWindow User Interface Language. The libraries are a collection of interaction tools such as menu, check box, scroll bars, etc. These libraries are provided to ease the programming burden and at the same time enhancing consistency among applications.

2. Dialogue Control Component
   This component handles the sequencing of events and interaction techniques. Current research has shown that it is possible to define the component using two dimensional, non-textual techniques, like what we do in using a window-based applications [29].

3. Analysis Component

   This component is used to study and evaluate the user interface after it has been created.

Not all three components are of specific interest to us. Because we are trying to let users of the system to define manipulative operations easily, the second component, Dialogue Control Component, would provide certain hints.

Most UIMS use a textual specification to define the Dialogue Control Component. Like the problem faced in authoring systems, users (in this case, the UI designers) are reluctant to learn a language with rigid syntax [30]. Garnet [29] is the first UIMS that allows users to create highly interactive interface using direct manipulation techniques. One goal of Garnet is to create user interface using techniques that is as easy as using the interface themselves.

In our project, similar techniques are used to define not only the user interface, but also visual objects. The properties of visual objects, which include static attributes (e.g. color, position) and dynamic attributes (e.g. reactions to events during run-time) would be defined interactively, without the use of a programming language.

2.2.3. Visual Programming

Visual programming refers to the use of meaningful graphic representations in the process of programming [36]. The graphical interface employed by most windowing system can be viewed as a kind of visual programming.

The human mind acquires visual information at a significantly higher rate than text. We can quickly extract features from a picture, which we can access in a two-dimensional fashion, while it needs longer to extract the central idea from text, which is limited to sequential access [33].

Current research in visual programming may be categorized into the followings [36] [11]:

1. Visualization of Data, Program or Software Design
   Programming in these areas makes use of traditional one-dimensional textual programming languages. Those are the end users who can benefit from the graphical representation of the data. Examples are INCENSE [27], Program Visualization [7] and GUIDE [24].

2. Visual Languages for Handling Visual Information
   In this category, languages are designed for the processing of visual information, such as pictures and images. The languages themselves are textual and usually used in the area of pictorial databases [10].

3. Visual Languages for Supporting Visual Interaction
   Examples in this category include HI-VISUAL [19] which supports user interactions via icons and Squeak [9] which supports user interactions via a sequence of user actions with multiple input devices. The languages themselves are textual.

4. Visual Languages for Actually Programming with Visual Expressions
   The languages in this category are two-dimensional, using symbols, such as diagrams, flowcharts and icons, to specify the procedures involved in a

program. PICT [17] uses conventional flowcharts to represent the program, programmers interact with the symbols within the flowcharts to create their programs. Rehearsal World [16] provides a stage metaphor to the teachers so that they can specify a program through the manipulation of different performers, in a non-textual way. The most interesting research is Peridot [28] which let UIMS designers to specify the static and dynamic behaviour of user interface objects in a totally two-dimensional, non-textual way. This concept is furthered extended in Garnet [29] where textual specification is completely abandoned.

In our system, visual programming techniques are not definitely required. But one can realize that in order to make the system more flexible, users must be able to extend the features, especially the presentation styles. Extending the system features using textual means (i.e. programming commands) is not feasible for novice users. Tools must be provided so that users can specify new presentation styles and new class of objects through visual interactions with familiar interaction objects (e.g. menu, dialogue box). This kind of visual programming should be categorized into the last one mentioned above.

In the current system, we are only concerned with techniques that are useful to our application area, but the model provided can be further extended to other visual object manipulation applications.

Chapter 3
User's Model

3.1. A Simple User's Model

Future users of the present system might be novice to computers from various fields. To make the system popular, the efforts needed to learn the provided functions must be minimized. It would be most welcomed by the users if they can use the present system to express their ideas. In order to achieve the user-friendliness goal, we must provide a model that appears simple to the users and will hide all the complex and tedious operations from them.

3.1.1. Object-Oriented [1]*Presentation Material*

From the users' point of view, a presentation involves a collection of materials, including text, pictures, diagrams, etc., and a collection of presentation operations, for examples, animation, enlarge/shrink, fade-in/out, and the changing of attributes.

In the real world, users get the materials ready on different pieces of paper, and arrange them on a master sheet. Different pieces of sheets can be stacked with those on the top obscuring the bottoms. Moreover, the stacking order can be rearranged at users' will. In our system, the concept of a collection of materials is

1 In this thesis, the term *Object Oriented* refers to object oriented graphics generally, which is concerned with the representations of the graphics primitives. Unless specified, this term does not refer to object oriented programming.

realized using object-oriented graphics. Each piece of material is represented by an object. Users integrate the objects by placing them at suitable locations. Objects can be stacked in any order. The user will find that the operation of the system closely resembles the real life. This is not feasible using pixel-based graphics because the information of objects, such as the stacking order, positions of objects, etc. are lost once they have been inputted. The systems of this kind only retain the color information of individual pixels.

Other than resembling the real world, the object-oriented representation makes modifications very easy. What the system stored for different objects (except bitmap) are their geometric attributes, but not their bitmap equivalent. For example, a Bezier curve is stored as four reference points, together with attributes like the line style, color, thickness, etc. When a user wants to modify an object, he/she modifies the attributes directly, not the bitmap displayed on the screen. So users are able to resize, move, and change the attributes of objects easily, while the system can display the results accurately.

Object-oriented representation of materials also allows users to duplicate objects easily. Moreover, the storage used by the materials are much less as compared to the pixel-based equivalent. The technical issues are discussed in section 4.3 in this thesis.

Object-oriented drawing packages are widely available nowadays. Examples are MacDraw and Micrografx Designer. Users who have experience in packages of these kinds should find no problem in preparing courseware in ours.

3.1.2.Frame-Based Presentation

Materials (objects) are organized and presented in a frame-by-frame basis. The size of a frame is equivalent to the size of the screen minus the area occupied by the title bar and the menu bar. Each frame resembles a slide used in a traditional presentation. Within a frame, objects are presented in sequence according to the presentation operations applied on them.

Some presentation package, like Authorware Professional, do not employ the frame concept, objects can step through the presentation from beginning to the end. This contradicts to the traditional approach, in which materials are brought to and removed from the presentation area slide by slide. It requires, moreover, the users to explicitly specify which objects are to be removed and when they are to be removed. In a well organized presentation, frames will help users to group their materials.

3.1.3.Presentation Styles

One major advantage of computer presentation over traditional one is the availability of different presentation styles. Earlier packages like Slide Presentation provides simple frame transition styles, such as fade-in/fade-out. Later ones allow users prepare more dynamic presentations by specifying the object movements.

In our system, the presentation styles are determined by the operations acting on the objects. The presentation operations apply on objects either sequentially or randomly. The moving of multiple titles in an arranged sequence from the edge of the screen to the centre is considered as sequential. The time when the blocks of objects are moved are well defined. Sequential operations are carried out in a specified sequence despite of the user's interactions.

Random operations are carried out only in response to the user's interactions. For example, the *ReverseColor* operation is applied to the object only when the user clicks the relevant item.

3.2. Novice Users vs Experienced Users

Our system will provide many basic objects such as text, geometric shapes, diagrams and will accept bitmaps from other applications; and will provide a number of basic manipulative operations for the objects. Novice users can use these system defined objects and operations to create an ordinary presentation.

For experienced users, the objects and operations provided may be inadequate. They are expected to define new object types and operations by themselves. For examples, a user who imports bitmap frequently would like to decorate each of the bitmaps with a rectangular frame, then a new object type (e.g. *FramedBitmap*) can be defined based on the old ones. New bitmaps with frames will then be created with lesser effort.

Similarly, new operations can be defined based on old ones. Users can combine the move and reverse color operations to produce a *MoveAndFlash* operation. More details will be given in section 5.5.

Chapter 4
Design of the Courseware Production and Presentation System

4.1. Overview

In the last chapter, a very general user's model is given. The overall design of our system, the available graphics primitives (or object classes), the available operations, and the concept of user-defined object classes and operations will be discussed in details in this chapter.

4.2. Object Oriented Design

Our system treats every visual object (e.g. line, text, picture) as a separate entity. Each of them has its own properties, including physical attributes and dynamic operations. Each type of objects has its own set of drawing commands.

Firstly, the purpose of using object oriented design is to match with the object oriented presentation material perceived by the users; secondly, new graphics primitives, or new types of objects created by the user can be integrated into the system because the design hide the low level display command of new objects from the high level control code.

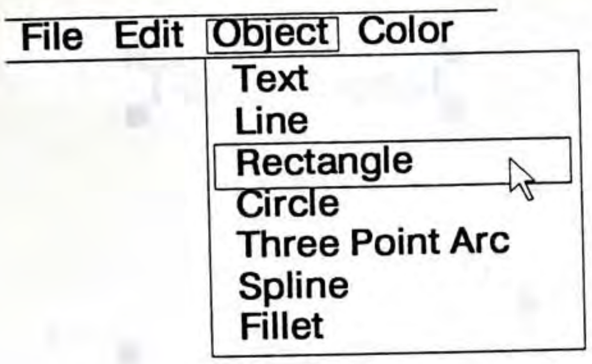
4.3. Object Oriented Graphics

In our system, the following graphics primitives are supported currently: text, straight line, rectangle, circle, Bezier curve, arc and fillet curve; and bitmap in the form of a TIFF (Tagged Information File Format) file can be imported.

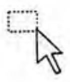
These graphics primitives and bitmaps are considered as object classes. A user can select the desired class from the menu and draw on the screen, that is, an instance is created. The instance object can be further modified. Figure 1 shows an example of creating and modifying a rectangle.

4.3.1. Modification of Object

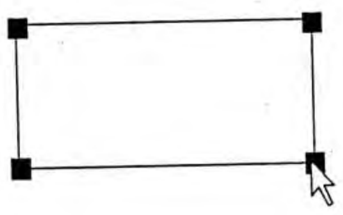
Representing each graphics primitive by storing its physical attributes, such as position, size and color, makes it possible to modify and duplicate the object. Moving an object is as easy as dragging it using the mouse pointer. Users can directly point to the handle of an object and drag it to the desired location, thus changing the size or shape of the object. Handles are visual representations of the control points of objects. Figure 2 shows some graphics primitives and their corresponding handles.



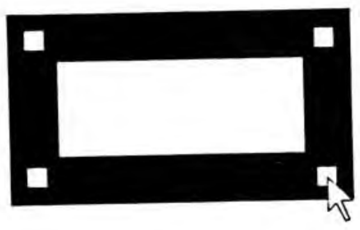
1. Select Rectangle from Menu



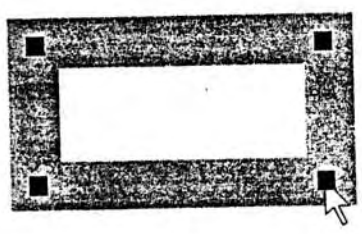
2. Press mouse button at the starting location, then drag mouse. Dynamic feedback is given.



3. Release mouse button at the other corner of the rectangle. The desired rectangle is drawn. The rectangle becomes active. Four handles are drawn. The size can be modified by press and drag on the handles.



4. Line width can be changed by selection from Width menu.



5. Line pattern can be changed by selection from Pattern menu.

Figure 1. Drawing and Modifying a Rectangle.

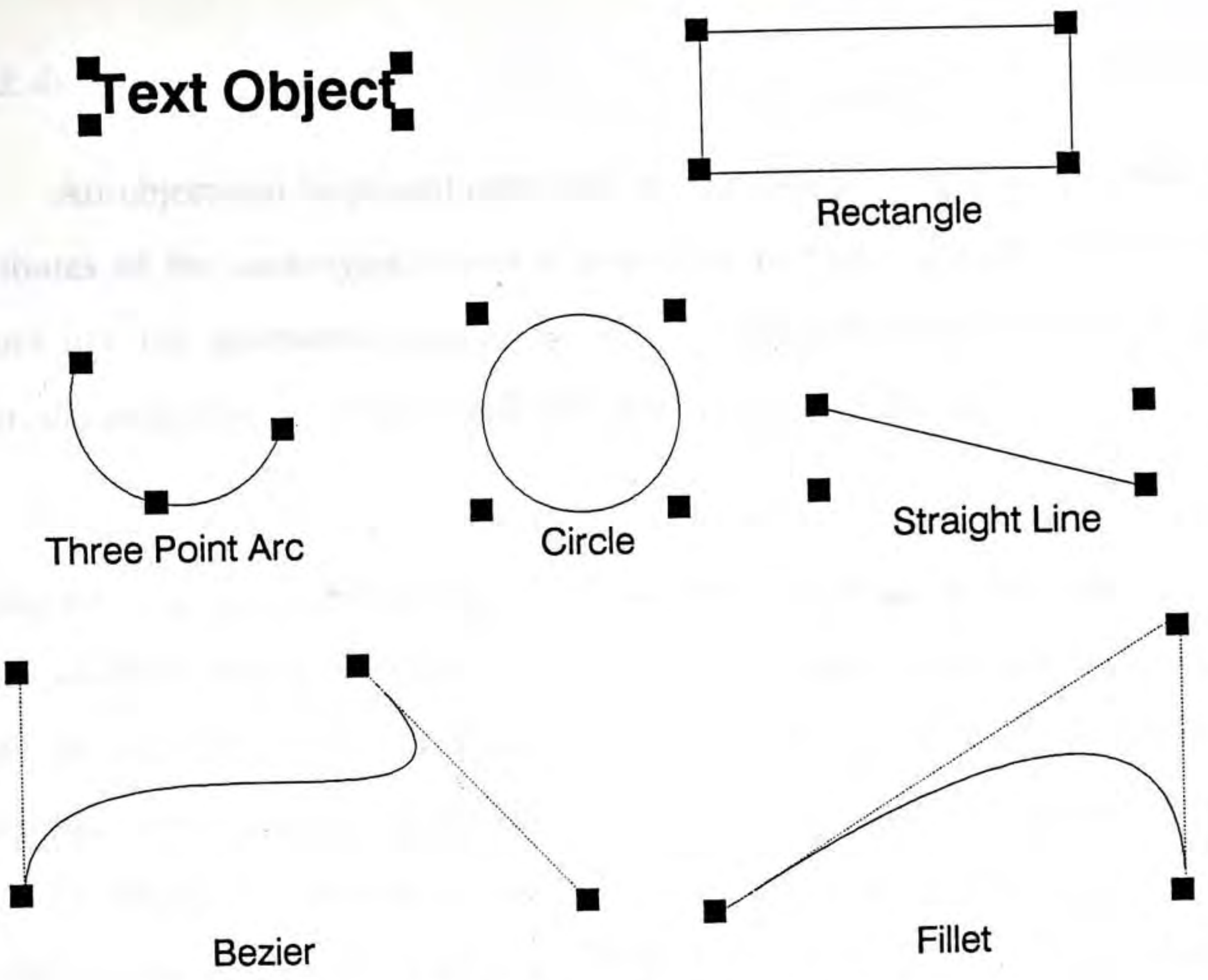


Figure 2. Handles of different objects.

Bitmaps (imported from TIFF files) cannot be further edited, but they can be resized and moved. Four handles at the four corner of the bitmap are provided for modifications.

4.3.2. Clipboard

Objects can be transferred from one editing window to another through the clipboard. The clipboard is a temporary area, which is invisible to the users, that can hold any object, one at a time. Users can select Cut and Paste from the menu to use this feature.

4.3.3. Stacking of Objects

An object can be placed over and obscure all or part of another one. The attributes of the underlying object is preserved because what we stored is the values like the geometric position. So if the overlapping object is moved away later, the underlying one can be redrawn without any difficulty.

The stacking order refers to the order in which the objects are to be displayed. The one displayed first might be obscured by those displayed later, if their positions coincide. Generally, those objects created later will be displayed later, on top of the others. It is nonsense to force the users to create the objects according to the stacking order. For this reason, two operations are provided to alter the stacking order. One is used to bring an object to the top while the other sends it to the bottom. No operation is provided to insert an object into a specific level, or between two specific objects, because the above two operations are enough to arrange the objects into any stacking order. Adding extra operations will complicate the system and this should be avoided.

Figure 3 shows a scenario that a bitmap and a text object are arranged in two different stacking orders.

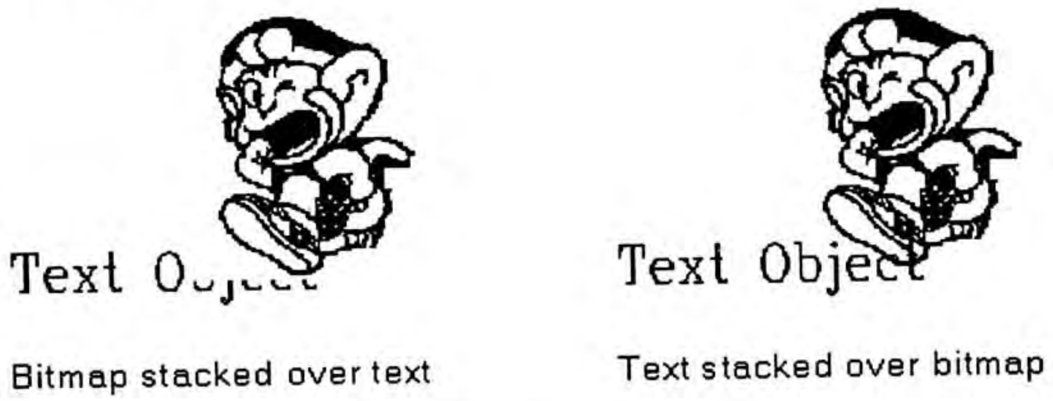


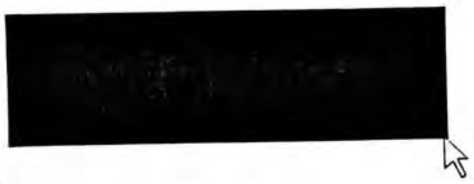
Figure 3. Stacking orders.

4.3.4. Group Together and Break Apart

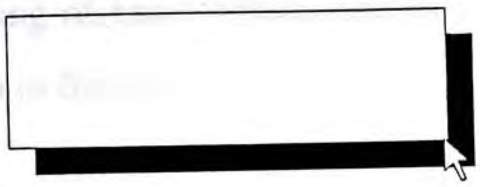
Several basic objects can be combined into a Group object, which is then manipulated like one single object. This is very useful if a user has edited several related objects and want to treat them as a single one afterwards. A Group object can be moved and its attributes can be changed. In the current system, a Group object cannot be resized for the sake of simple implementation. This can be achieved by first storing the position of each object relative to a corner of the Group object, after the Group object is resized, all the components will be resized proportionally.

To group several inter-related objects together, one must first select all of them. Figure 4 shows an example of grouping a heading and two rectangles which form a shadowed backplane together.

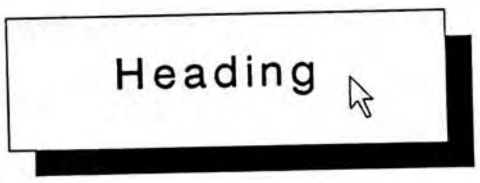
A Group object can be broken apart into its original components later by the *Break* operation in the *Edit* menu.



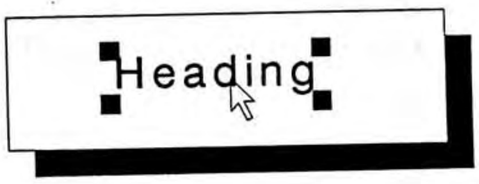
1. Draw the shadow first.



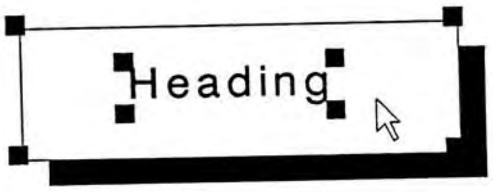
2. Then draw the back plane.



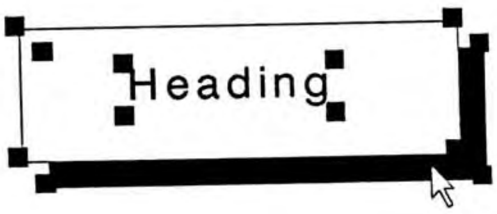
3. Type the heading text.



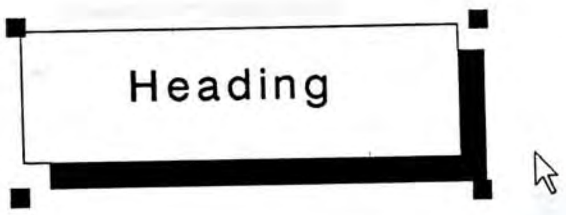
4. Click on the text object to select it.



5. Click on the back plane while pressing the Shift key to do multiple selection.



6. Similar to last step, select the shadow also.



7. Select Group from the Edit menu, the three objects will be grouped together. Only the handles of the Group object will be shown. The three objects can be moved and modified as one single object then.

Figure 4. Multiple selection for grouping objects.

4.3.5. Hierarchy of Grouping

A Group object can consist of basic objects or Group objects. Further grouping of basic objects and Group objects will form a hierarchy like the one shown in figure 5.

By looking at the hierarchy, one can easily realize that breaking a Group object will only affect the highest level in the hierarchy, those at the lower level will remain intact. This is logical because it is the user's will to organize the objects in that way.

The group-and-break mechanism will be used again when a user want to create a new object class. This will be discussed in section 5.4.

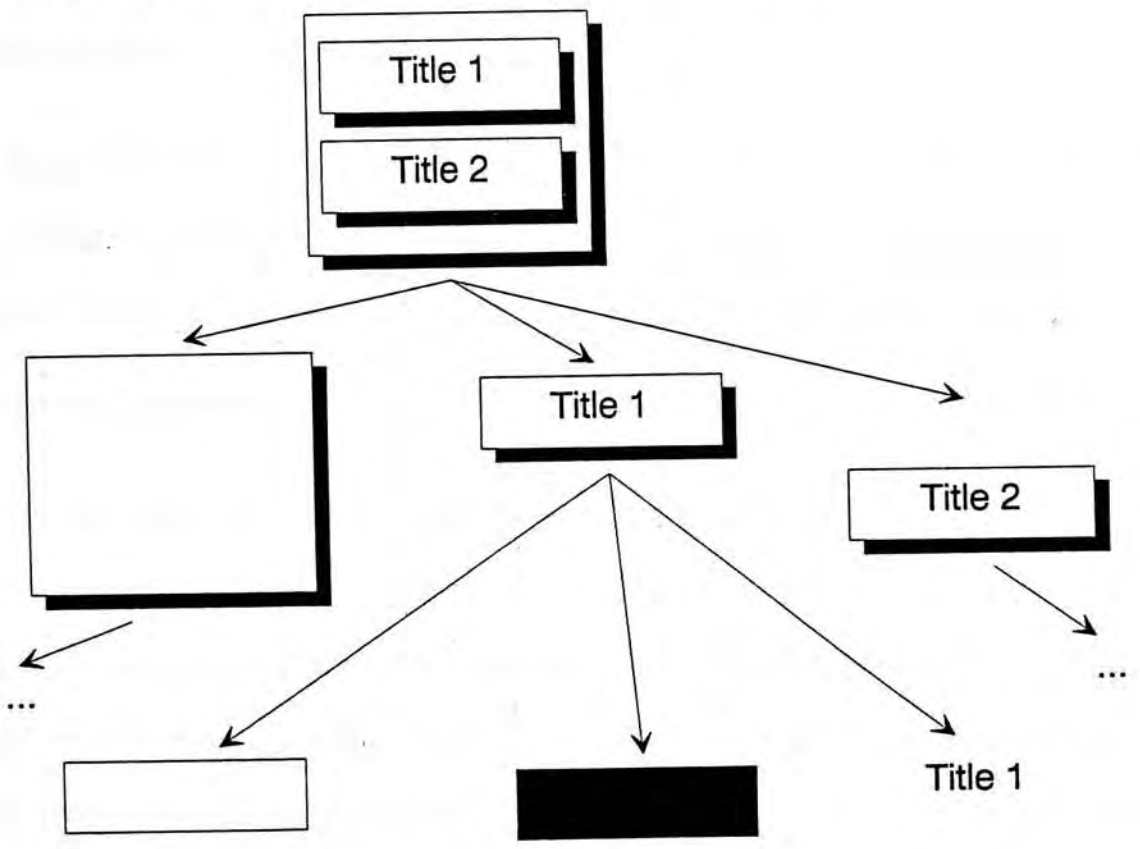


Figure 5. Grouping hierarchy.

4.3.6. Storage Requirements

Less memory space is required by object-oriented representation as compared with bitmap representation. This is also true when the objects are saved on to the disk. About sixty bytes are required to hold the information of a straight line such as the coordinates of the starting and ending points, line thickness, pattern, color, etc. The straight line can then be drawn at any size and slope. This is impossible if bitmap representation is saved.

4.4. Operations

In our design, there are three types of operations that could act on the objects: manipulative operations, frame control operations and timer operations.

4.4.1. Manipulative Operations

Examples of manipulative operations include move, show, hide, change color, change thickness, rotate, etc. These manipulative operations could be initiated through direct manipulations, or simple selections from menu bar.

4.4.2. Frame Control Operations

These operations are used to select the frame to be displayed, which include next frame, previous frame, go to frame N, first frame and last frame. Recall that the presentation is frame-based, and the size of a frame is limited by the size of a screen. The frame control operations will allow the user to switch quickly from one frame to another. They are also very useful when bound to objects like buttons. In this case, a user can specify which frame is to be jumped to when a button is being clicked. The binding of operations to objects will be discussed later in chapter 5.

4.4.3. Timer Operation

As the operations described above are all immediate actions, a timer operation is sometimes needed to achieve dynamic actions such as flashing.

The timer operation provided is to set an alarm at which an action will be activated. This is similar to the *delay()* system call in UNIX [2] in the sense that the object associated with this operation goes to sleep (i.e. idle) for a period before another operation is performed.

The flashing example can be defined as follow, where the color of an object will be reversed *Freq* times in a second:

```
Flash(Freq, Object) := Timer( 1/Freq, Reverse&SetNextTime ( Freq, Object) )
Reverse&SetNextTime(Freq, Object) := ReverseColor(Object), Flash(Freq, Object)
```

The timer operation cannot be activated through direct manipulation. This is logical because all operations performed through direct manipulations are immediate. The timer operation will be used only in defining user-defined operations. The definition of new operations will be discussed in chapter 5.

4.5. Active-Object-Set Model

In the last two sections, a general concept of objects and operations are provided. Here, in this section, the underlying relationship between objects and operations are discussed. An Active-Object-Set Model is proposed to provide a simple view on this relationship.

4.5.1. Importance of Objects

In the theatre metaphor proposed in the Programming by Rehearsal system [16], every visible object in a program is described as a performer, each performer affects one another by sending cues. By using such a metaphor, the unfamiliar concept of programming can be conveyed to non-programmers rather easily. This metaphor is, in fact, the object-oriented approach to manipulate visual objects. The performers (i.e. the objects) affect others (i.e. activate operations that act on other objects) when certain events occur, or at certain pre-set time. Linton, Vlissides and Calder [8] suggested that the code for user interface should be object-oriented for easier development and maintenance, and MacApp [34], which is a generic Macintosh application, is also object-oriented. Objects are natural for representing the elements of a user interface. This is the users' viewpoint on the application, and it should be the same for programmers' viewpoint.

The proposed Active-Object-Set Model simplifies the relations between objects. And it enables the binding of events and call-back operations to objects.

4.5.2. Active Object

An active object is the object currently under selection and has the input focus[1]. Our system can be viewed as a collection of various visible objects, with one of them being the active object. The system will transit from one state to another when the state of individual object changes. The state of an object may be changed by either external events (e.g. mouse click, timer) or internal events (invoked by other objects). Figure 6 shows a generalized example of the state transition of an active object. It should be noted that not only the active object itself is modified, but the related objects will also be modified.

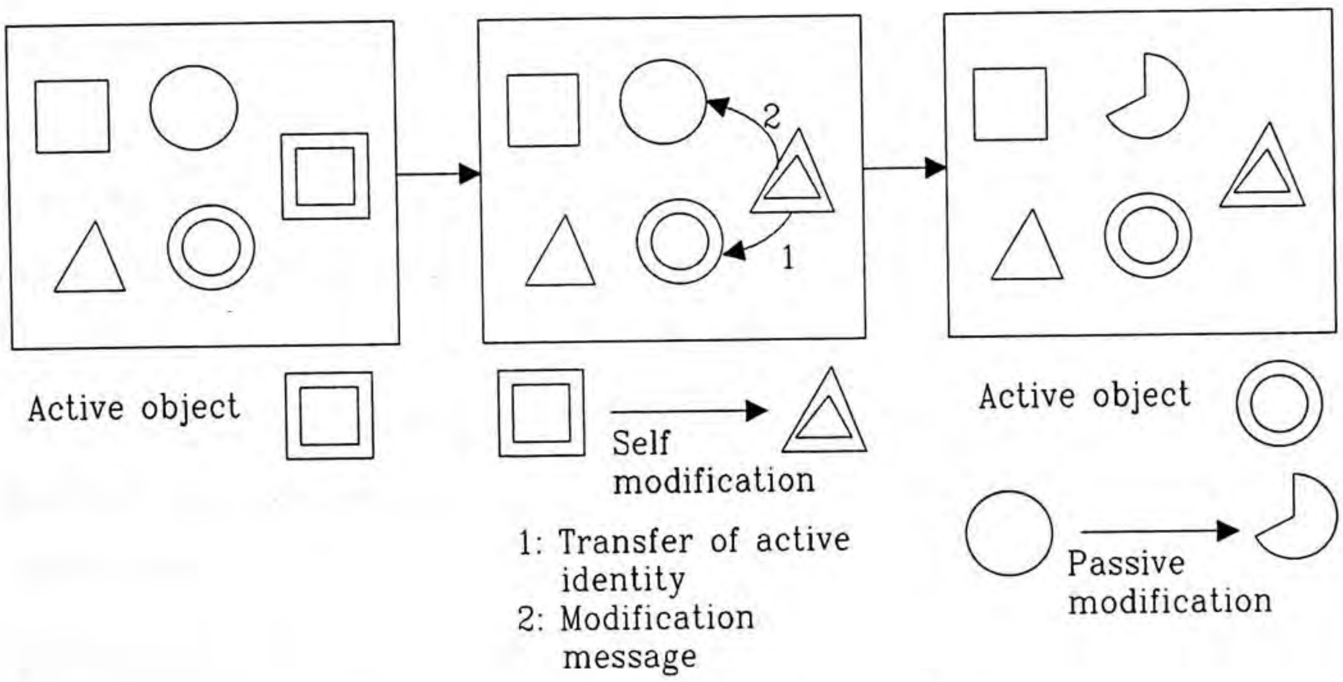


Figure 6. State transition of an object.

1. In OS/2 Presentation Manager, the window that is expecting input from a mouse or a keyboard is said to have the 'Input Focus' [31]. Events generated from the two input devices are sent to the window with the input focus. In this report, however, an object with the input focus means that all the input from the user are directed to that object.

Any object can become active by i) an external event (e.g. mouse click); ii) identity transfer from previously active object. This concept is further extended to include a set of active objects and to allow non-active objects to respond to the event generated by the system clock.

4.5.3. Active Set

On many occasions, users may want to perform an operation on a group of objects simultaneously (e.g. move/delete a group of objects). An active set, rather than one active object, should be used to describe these situations. In another model [15], the active object is referred to as the currently selected object (CSO), similarly, this concept can be extended to a currently selected set of objects. Only those events that are common to all members in the active set could be processed, not otherwise. For example, mouse movement with button pressed is a common event for a group of objects, which may be of different types. Some menu items, such as the selection of typeface, size, styles, etc. will not function (except for the case that all members in the active set are textual objects) because such a selection is meaningful only to textual object. The active set is realized by allowing a user to perform multiple selections, and perform operations on the selected set afterwards.

4.5.4. The Timer Event

As [15] suggested, typical types of events that can be placed in the event queue can be classified into 1) events generated by interaction devices (e.g. keyboard, mouse and lightpen); 2) events generated by the window manager because of changes in output window status (e.g. window exposed or resized) and 3) event generated by the system timer. The second type of events are handled by a default action, which involves the repainting of the output window, and will not be sent to the active object. All other events are directed to the active object. Non-active objects would be modified only if they are commanded by the active object. This is acceptable to the events originated from interaction devices because the active object is assumed to have the input focus. The timer event, however, may not be received by the active object. It can be any object that asked the timer to send an event at a specific time. It may not be the active set who is interested in the event. So system timer event should be allowed to be processed directly by the target object, not the active set.

The revised event-handling diagram is shown in figure 7.

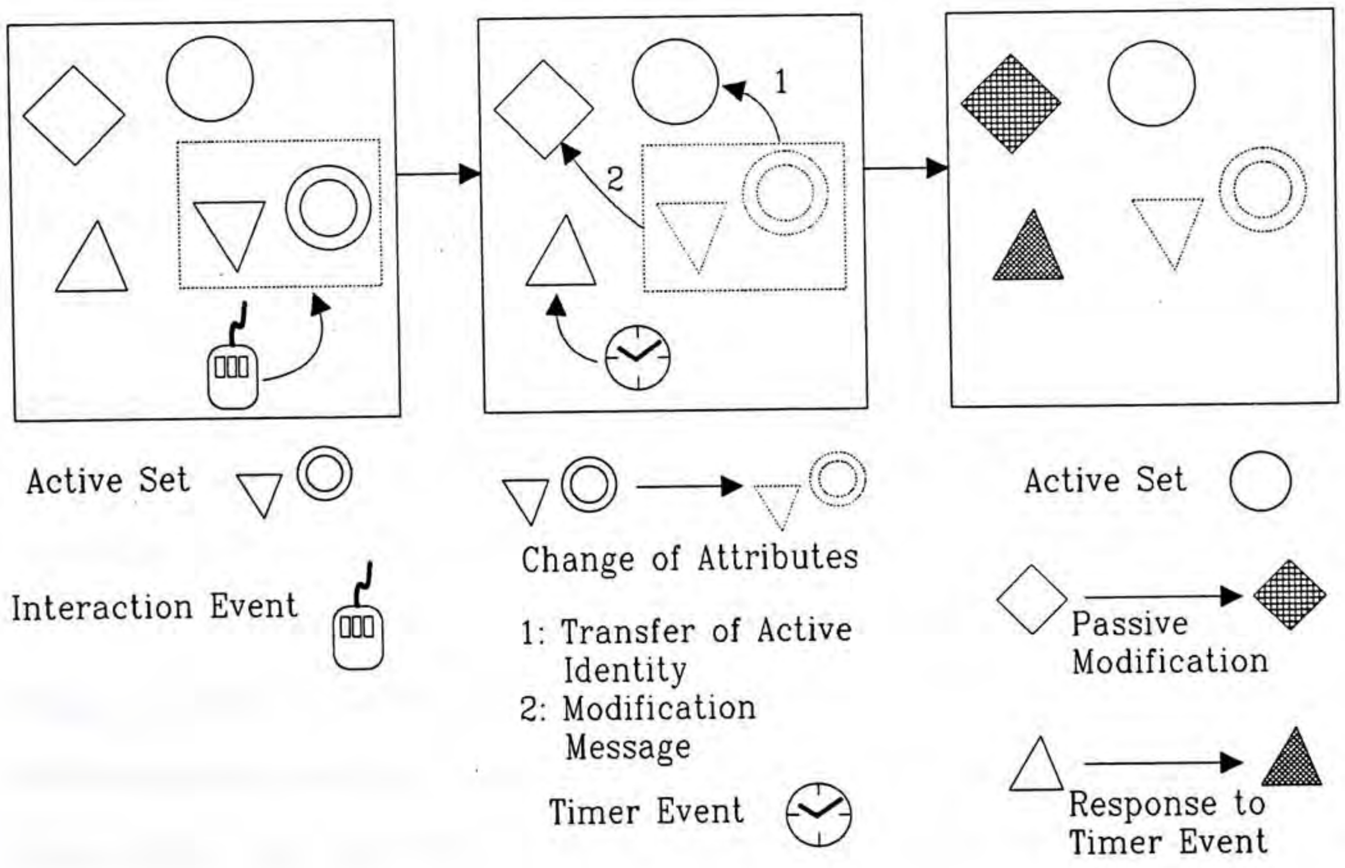


Figure 7. Event-handling of objects.

Back to the flashing operation example, after an object initiates the Flash operation, the system will send the timer event to the object after a certain time. That object may not be active at the time it receives the timer event. This enables an object flashing while other objects are active. If the timer event is, like other events from keyboard and mouse, sent to the active set, the above example is not possible.

4.6. Properties of Visual Objects

4.6.1.Physical Attributes

Every visual object has its associated set of physical attributes. General attributes are color, size, position, etc. Attributes specific to a certain class of objects are font and style for textual objects, line width and line style for geometric lines, pattern for geometric areas, etc.

4.6.2. Event-Handling Operations

The properties of visual objects refer to more than just the set of physical attributes. They include the operations associated with the events which will be directed to the visual objects. For example, in an event-driven environment, an operation will be launched when an event occurs, while the event will be directed to the active set. So it is sensible to bind operations to events. Moreover, each class of objects processes the events in a different manner, it turns out to bind different event-handling operations to each class of objects. X Windows System has a similar approach by associating callback functions to the classes of widgets [37].

4.6.3. Private Status

In an event-driven environment, a complete command may involve several events, which are in sequential order. For example, the command to move an icon from one location to another involves the following sequence of events:

1. Mouse button pressed with pointer located at the icon;
2. Mouse movement with button pressed; and
3. Mouse button released.

The icon, driven by events, transits from one state to another. Some private status flag are needed to maintain such information. This issue will be further discussed in section 7.1.6.

The properties of a visual object is visualized in figure 8.

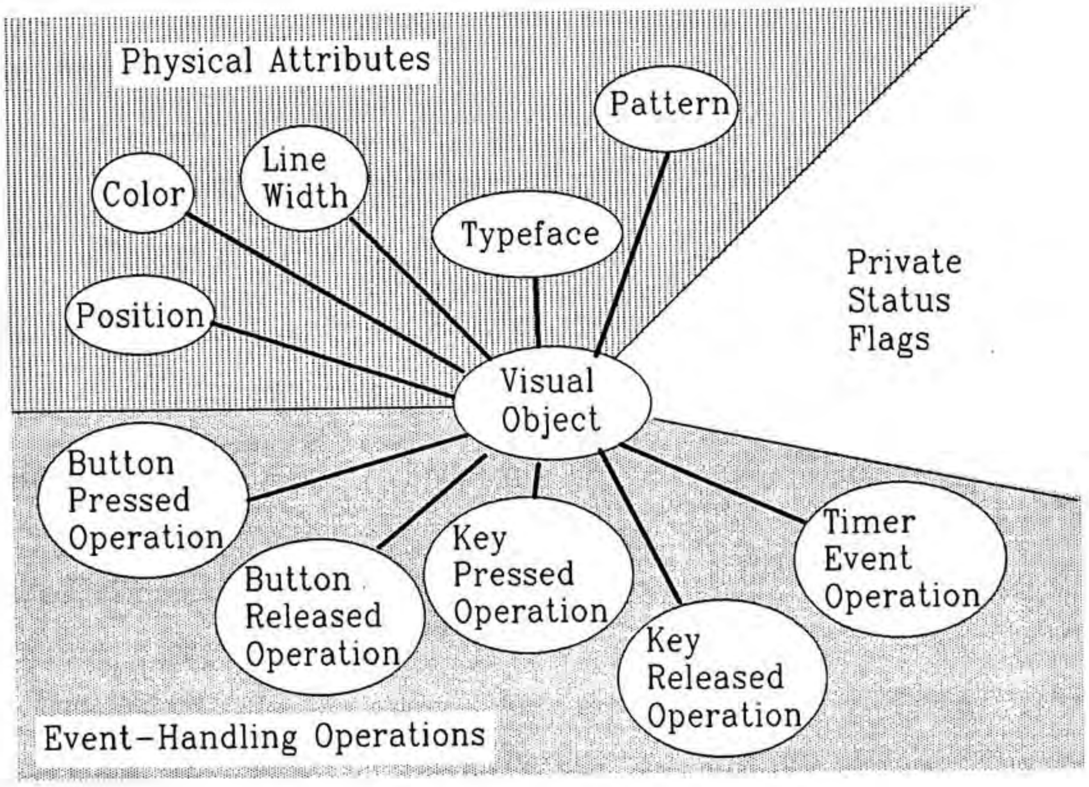


Figure 8. Properties of visual objects.

4.7. Object Class

Each graphics primitive provided by the system is a basic object class. Each object class has a set of attributes defining the objects belong to that class (i.e. the instances). For example, the Rectangle class consists of attributes like coordinates of the upper-left and lower-right corner; thickness, width, color and pattern. An instance of the Rectangle class has its own values quantifying the attributes, i.e. different object instances can have different attribute values. One rectangle can be red while the other is blue. When defining a new object instance, all the attributes of that instance will be filled with default values. (An exception is that the position and size of geometric objects are specified through direct manipulations, which conform to the specifications used in popular drawing programs. Similarly, the characters inside a Text object is filled in at the time the user types them, no default value is given.)

4.8. User-Defined Object Classes

The basic objects provided by the system may not be enough for the users. Experienced users might want to create some customized shapes and reuse them frequently. One example is to decorate the border of another object. By defining the border as a new object class, one can create instances of it conveniently, and the appearance of the border can be maintained easily.

4.9. User-Defined Operations

The system will provide a basic set of primitive operations. The user, if required, can use them to create some desired operations. The concept of combining primitive operations into a complex one is similar to that of combining primitive objects into a complex one. The former addresses to the dynamic portion of the system while the latter addresses to the static part. In the present system, the operations provided are manipulative actions, such as to change the color of an object, to move or rotate an object along a certain path.

More details of defining new object classes and operations will be given in the next chapter.

Chapter 5
Interaction Techniques for Defining New Object Classes and Operations

5.1. Interaction Techniques

In this thesis, the term *Interaction Techniques* is used to refer to the use of physical interaction devices, such as mouse and keyboard, to perform actions that are then translated to different commands or values. In a graphical, windowed environment, the physical devices are abstracted using interaction objects like menu, scroll bar, button, etc.

This chapter will concentrate on the discussion of how users can interact with the system to define new object classes and operations.

5.2. Object Creation

To create a new object, that is, to make a new instance of a new object class, the user should choose the appropriate item from the Object menu. For the Text object class, the user can then click on the desired starting location and begin to type in the text. For other objects, rubber banding [15] is used to define the appearance of the object. Users familiar with MacDraw like applications will find no difficulties in using them.

Ideally, there should be a graphical toolbox for the editing window, listing all the available object classes. Users can then select the appropriate tools more conveniently in the sense that little mouse movement is needed. And because the toolbox will be always visible, one can quickly locate which object class is needed and move to it quickly.

An object can be further modified by first selecting it, making it the active object. It can be done by simply click on it. The activated object will have handles surrounding it. If several objects are stacked together, clicking on them may select only the bottom one. The others can be done by double-click on them, then the objects on top will be selected successfully.

Each object has a bounding rectangle, the object is selected when the user clicks inside the object's bounding rectangle. Sometimes this brings confusion. Figure 9 shows an object that is 'occupying' the whole area, making selections difficult. The user needs to double-click all the time to select the desired one.

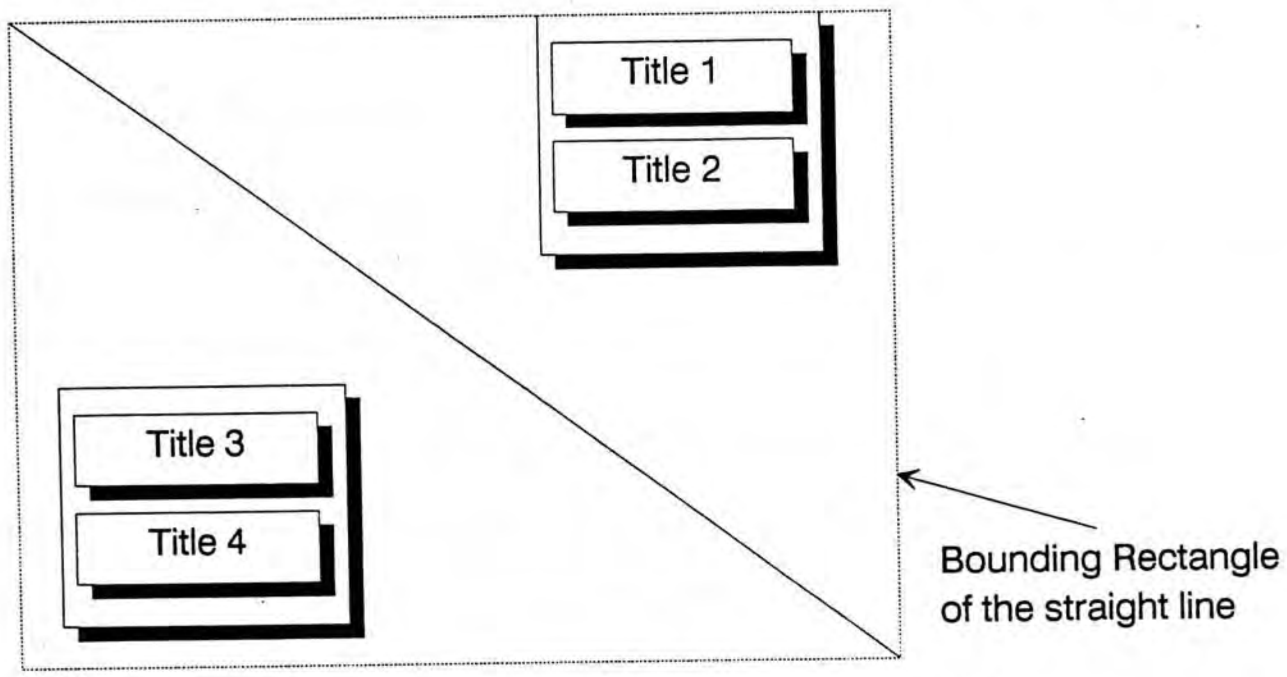


Figure 9. Object with large bounding rectangle.

5.3. Operations

The operations that act on objects can be activated either by direct manipulations or menu selections.

5.3.1. Direct Manipulation

The most simple example of direct manipulation is the Move operation. To move an object, the user first selects it by clicking, then the object can be moved by dragging. (Dragging means to move the mouse pointer while the mouse button is being pressed.) Other operations like resize and reshape can also be done through direct manipulations. This gives the user a feeling of moving or reshaping the object directly with his/her own hands.

5.3.2. Menu Selection

Other operations are not so easy to be manipulated directly. They are generally activated by selecting a command from the menu. For example, to change the color of an object is done by selecting that object first, then the desired color is chosen from the menu.

In fact, some commands that are activated by menu selections can be replaced by direct manipulations. This, however, might not be as convenient as menu selections. In section 7.1, we will further discuss the issue.

5.3.3. Parameter Selection

Operations need operands to act on. They are functions or procedures from the view point of programming languages; and the operands are parameters passed to the operations.

Postfix notation is used in specifying parameters, that is, the operands needed are specified first before the operation is selected. An object to be modified is selected first before any operation is performed. Some operations might accept many parameters; examples are move, cut, copy, and change color. Multiple parameters can be specified by multiple selections. The techniques of multiple selection are described in section 4.3.4.

If no object is selected, direct manipulation cannot be performed. Operations like move and reshape will be restricted from the user's access automatically. Other operations that are activated through menu selections will still be available to users. In our system, these operations will affect the default values of the objects. For example, if no object is selected, the selection of Red from the Color menu will affect the default color, all objects created later will be red in color.

If some operations that are activated through menu selection need at least one parameter, then that menu item can be deactivated (or greyed) when there is no active object, so that the user is barred from making the invalid selection.

5.4. New Object Class Definition

In our design, a new object class is created from aggregating the basic classes. This is similar to the RECORD in PASCAL and STRUCT in C which allow programmers to define abstract data types based on predefined data types. The user can make instances of the new class as he/she does for the basic object classes.

5.4.1. Definition through Drawing

To define a new object class, a prototype will be drawn. The prototype drawn will consist of several basic or user-defined objects. They will be grouped together and declared as a new object class. A new object class name will also be input for identification.

For example, a user can create a new object class called *ShadowedButton*. A prototype of the class is shown in figure 10. It consists of two rectangles and a text object. The string 'Button' is just a sample text. They are then grouped and declared as *ShadowedButton*. Similarly, in figure 11, a prototype for the class *FramedBitmap* is drawn. The system will save the sizes, positions, line styles, thickness, color, pattern, etc. for the created object.

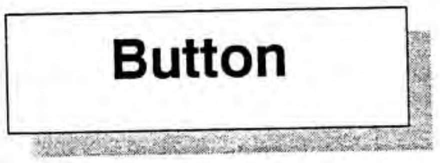


Figure 10. The *ShadowedButton* object class.

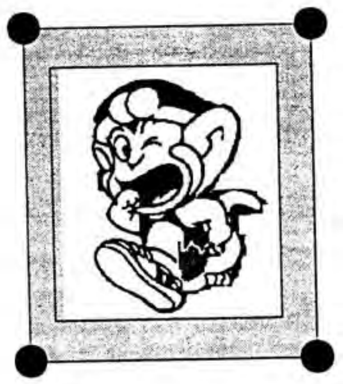


Figure 11. The *FramedBitmap* object class.

For the *ShadowedButton* object class, the font and size, and the style of the sample text will also be saved, but the string 'Button' will not because it is not as meaningful as the other attributes. The user would properly override this value in future use. For the *FramedBitmap* object class, only the relative position would be saved. This is similar to the text object of the *ShadowedButton* class that the content should be filled in by the user when a new object instance is created.

5.4.2. Creating New Object Instances of the New Object Classes

Using the new object class will be as simple as the basic ones. The user will first select the right tool from the menu. The new object classes could not be selected from the menu directly because they are not supposed to be put on the menu at first. They will be chosen indirectly through a dialogue box, as shown in figure 12. When a new object class is selected, the user will then specify the coordinates of the bounding rectangle of the new object through direct manipulation, just like drawing a rectangle. The stored prototype of the new object class will be scaled to fit in that rectangle and displayed.

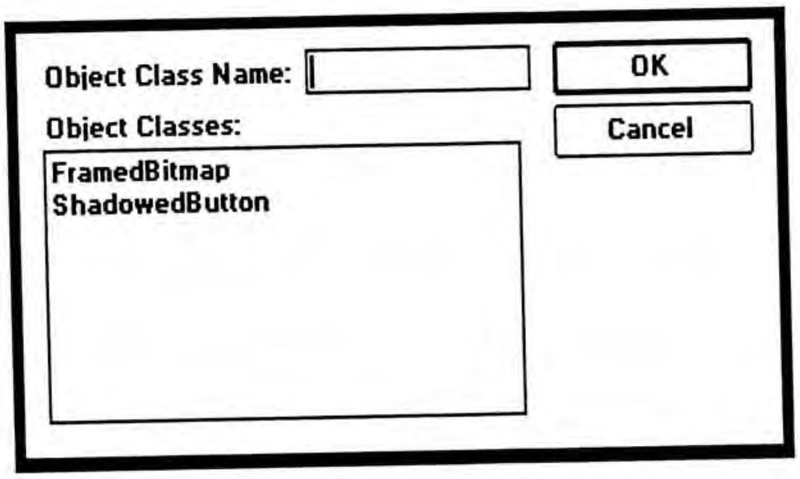


Figure 12. Dialogue box for choosing user-defined class.

Back to the example of *ShadowedButton* and *FramedBitmap*, two ingredients are to be input at creation time, that is, the text and the bitmap. The text object can be input as usual. A dialogue box will be poped up for the selection of bitmap files from the disk. This would be the same dialogue box that will appear when the user choose 'Load TIFF ...' from File menu. The bitmap loaded will be displayed within the frame of the *FramedBitmap* object then.

Up to now, the user-defined object classes can be viewed as library objects because only the physical attributes of the objects are saved. When the user wants to make a new instance of a class, the stored object will be scaled and displayed. The attributes like line style, width, color and font will be the same as those stored, but not depend on the current default values. Later in this chapter, the reader will find that more than just the prototype will be saved for a user-defined object class. Operations will be associated with object classes also.

5.5. New Operations Definition

The concept of combining primitive operations into a user-defined operation is similar to that of combining primitive objects into an integrated one. The former addresses to the dynamic portion of the system while the latter addresses to the static part.

There are well-developed interaction techniques for the physical attributes for the user-created objects, but not so for the user-defined operations. The following issues must be considered in order to make the definition of user-defined operations via visual interactions possible:

5.5.1. Specification of Parameter Type

Operations accept parameters which are objects that the operations will act on. The user is required to specify the types of these parameters. Visual interactions can be used in the specification process. For example, if a new operation is to accept a text object as the parameter, the user can specify it by selecting a sample text object with the mouse. This selection shows the system just the type of the *formal parameter* of the new operation, not the *actual* one, otherwise, the text object will become a constant in the operation.

Specifying the type of the parameter is very important. An operation will function properly only when the right type of object is chosen. This is not obvious if the parameter is a basic object, such as *Line*, *Rectangle*, or *Text*. For these basic objects, if the operation is not applicable to them, some checking routines can be built-in to prevent any run-time error. The simplest way is to ignore the operation. One example is applying the *Change Font* operation on a *Line* object. It is difficult, however, for the system to handle the error for a user-defined operation acting on a user-defined object. A user-defined operation may consist of a sub-operation[1] that act on a sub-object[2] of the user-defined object. If an object of invalid class is selected as the parameter, the system will not be able to find the sub-object.

Sometimes an operation can act on several classes of objects. The user should have the opportunity to give this information explicitly. For example, if a user wants to define an operation that act on rectangular shape object, that is, *Rectangle, Bitmap* and *Text*, then he/she can:

1. Select a sample object first, the sample object can be any one of the desired object classes;

1 One can see that a user-defined operation is actually a sequence of other predefined operations (i.e. sub-operations).

2 Recall that a user-defined object class is an aggregation of primitive objects, a sub-object is one of those primitive objects.

2. Select *Define Operation* from the menu, a dialogue box similar to the one shown in figure 13 will appear. The *Parameter Type* box lists all the available object classes, with the one the sample object belongs to highlighted;
3. Highlight also the other classes in the list box that can be the parameter.

By doing simple selections, the system will then know clearly what kind of parameters will be passed to the operation.

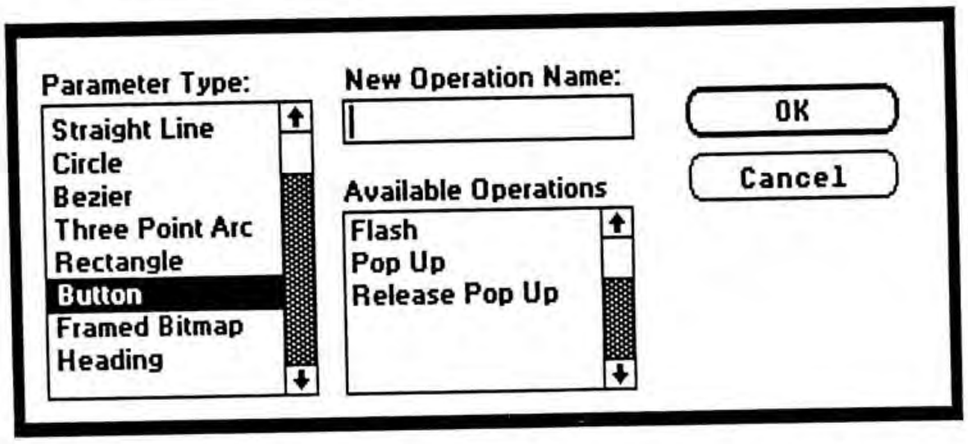


Figure 13. Dialogue box for defining new operations.

5.5.2. Selection and Sequencing of Primitive Operations

The new operation is composed of a sequence of primitive actions. Similar to keystroke macros, the sequence of the primitive operations would be demonstrated one by one by the user, and will be automatically recorded by the system. The demonstration will involve only normal interactions, which the user is already familiar with, that used to manipulate the objects.

If the parameter is a user-defined object, then the new operation can contain sub-operations that act on sub-objects of the user-defined object. To distinguish the currently selected sub-object from the user-defined object (i.e. the parameter), different types of handles will be displayed. Figure 14 shows a *ShadowedButton* with its sub-object, the shadow, selected. The user can select different sub-objects within the user-defined object, and then perform operations on them.

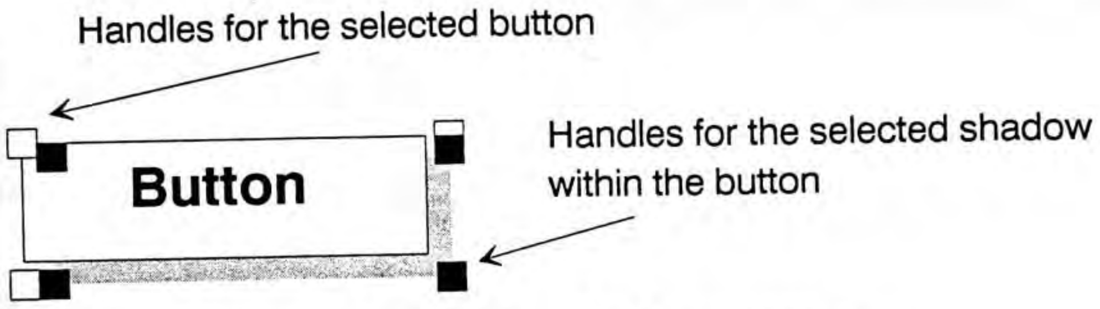

Handles for the selected button
Button
Handles for the selected shadow within the button

Figure 14. Selection of sub-components.

For example, if the user wants to define a *PressButton* operation, which acts on a *ShadowedButton*, that moves the text plane to the position of the shadow, as shown in figure 15, he/she can:

1. Select the button object first;
2. Select *Define Operation* from menu;
3. Select the upper *Rectangle* sub-object;
4. Move it to overlap the shadow;
5. Select the *Text* sub-object;
6. Move it back to the center of the upper rectangle;
7. End the definition process.

The process is shown in figure 16.

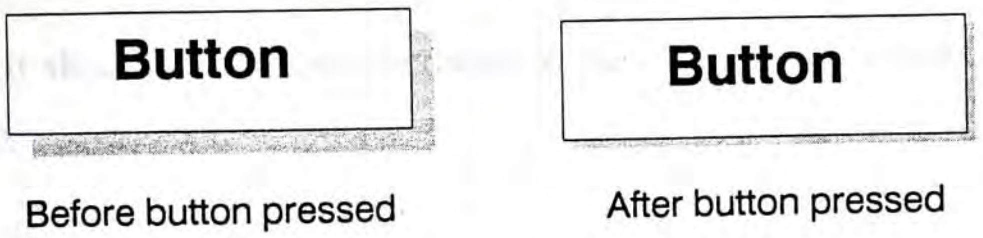


Figure 15. Button before and after pressed

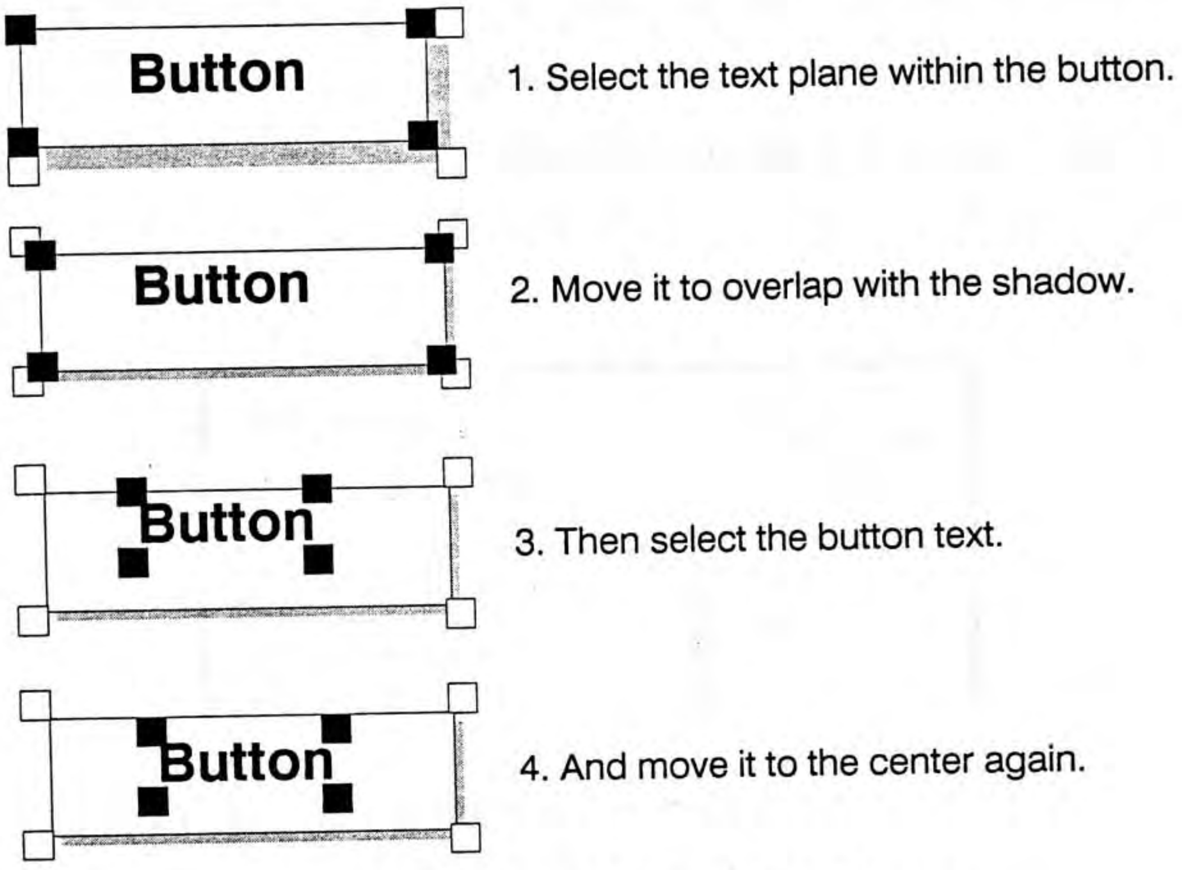

Button
1. Select the text plane within the button.
Button
2. Move it to overlap with the shadow.
Button
3. Then select the button text.
Button
4. And move it to the center again.

Figure 16. Sequence of *ButtonPress* definition.

A *ReleaseButton* operation can be defined similarly by moving the two objects back to their original positions. A user-defined operation can consist of other user-defined operations. The *Timer* operation can be integrated into the sequence also. Because the *Timer* operation will activate another operation at a later time, it should be the last operation in the sequence, otherwise confusion may occurs[3].

5.5.3. Using the New Operations

The newly defined operations cannot be activated from top level menu selection because they are not supposed to be in the menu at first. Through some indirection, however, they can be accessed by selecting *PerformOperation* from the menu. A dialogue box will appear to let the user to select the desired operation (figure 17).

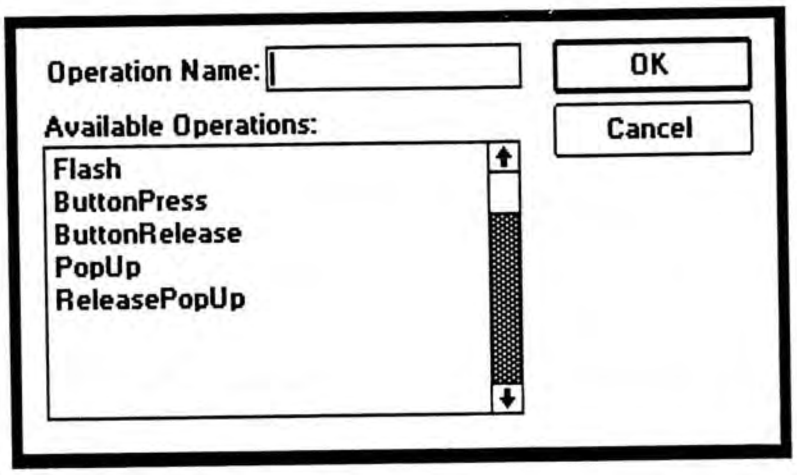


Figure 17. Dialog box for selecting user-defined operations.

Activating the user-defined operations through the above dialogue box is of little use. These operations are supposed to be activated through direct manipulations. Before they can be performed through direct manipulations, they

3 If another operation *A* is placed after the *Timer* operation, which sets operation *B* to be activated after some time, the user might expect that *A* would be carried out after *B*. In fact, *A* would be activated immediately after the *Timer* is set, and *B* would be activated at the preset time, which is later than *A*.

must be bound with events that will happens on objects. Next section will be dedicated to the discussion of this issue.

5.6. Binding of Operations to an Object

The last step of defining visual object properties is to bind the operations to the objects. These operations, according to the event-handling diagram (figure 7), may modify the object itself or other objects, that is, any object can be the parameter of the operation, not only the active object. The binding step must be able to indicate what argument will be passed to the event-handling operation once an event occurs. (The parameters specified during the operation definition phase described in section 5.5 just show the type of the argument that would be passed, the actual argument that would be passed in response to an event is specified here, in the binding phase.)

Operations will be activated when some events happen on an object. For example, the user might want an object to be highlighted when being clicked, the *MouseReleased* event should then be bound with the *ReverseColor* operation for that object. If a *MousePressed* event is expected to cause a *ButtonPressed* operation acts on a *Button* object, the *MousePressed* event should be bound with the *ButtonPress* operation for that *Button* object. A sequence of operations can be bound to an event.

In our design, the user will first activate the desired object, then select the desired event, and finally bind it with some operations. For example, we should select the *Button* object first, then activate the *Event-Operation Binding* dialogue box, as shown in figure 18. The desired event, the *MousePressed* event, will then be chosen. Next, the operation to be associated, i.e. the *ButtonPress* operation, to this event will be activated normally. The sequence of actions will be ended by clicking the *End* button in the dialogue box. The whole process will be the same as what should be done in specifying user-defined operations.

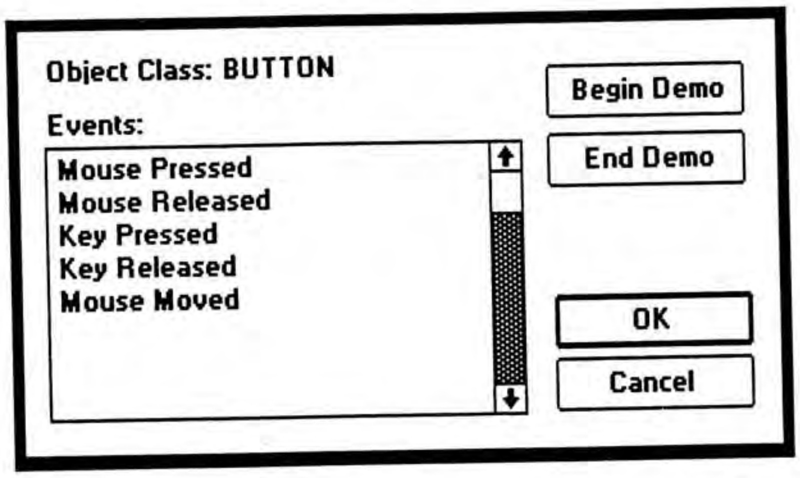


Figure 18. Dialog box for selecting the event and begin demonstration.

Assume the object under binding is *A*. The sequence of operations may involve the modification of other objects. The user can specify this by selecting and modifying other objects as usual. The system will treat the object *A* as the active object all the time, and the identity of the objects affected will be saved. When the sequence of operations are activated during run-time, the same (group of) objects are affected the same way as they did, while object *A* will remain the active object.

The sequence of actions may involve the transfer of activeness, that is, other objects can be activated by the current object. The transfer of activeness must be arranged at the end of the whole sequence of operations, or otherwise, the operations following will act on the newly activated object, which is not correct.

5.7. Default Operations for User-Defined Classes

In the last section, operations are bound to events that will occur to an object *instance*, that is, only the selected *Button* will carry out the *ButtonPress* operation in response to the *MousePressed* event. This is useful when we want to define specific actions for a specific object. But for the last example, we would rather have all objects of the *Button* class to carry out the *ButtonPress* operation in response to the *MousePressed* event. This should be the default action for the *Button* class.

To save a set of *Event-Operation* binding information as default values for a class will be very easy under current design. The user can select the prototype that, apart from having the desired physical attributes for the class, contains the desired *Event-Operation* binding information and save it as a class definition.. When a prototype is saved, not only the physical attributes will be saved, but also the dynamic attributes will be also.

Chapter 6
Implementation Issues

This chapter is devoted to the discussion of the implementation details of the Courseware Production and Presentation System.

6.1. Operating Environment

When we choose the operating environment for the system, the following points were considered:

6.1.1. The User Interface

The most important consideration of the system concerns user friendliness. The graphical user interface employed by most windowing systems is obviously the preferred choice for the following reasons:

1. Consistent User Interface

   Users will interact with the windowing environment through selections from pull-down menus, dialogue boxes and manipulations of icons, scroll bars, buttons, etc. Once user learned how to interact with a control, say, a button, he/she will know what can be done on this kind of controls later, no matter he/she is using the same application, different applications among the same system, or even different applications on different systems. The consistency is even more apparent among different applications within the same computer system. In the Macintosh environment, for example, there is always a

FILE menu for each application, allowing users to do file operations such as load and save. User interface consistency greatly shortens the time taken for users to adapt to a new application, and support transfer of users' learning [4]. As the present system will use interaction objects such as menus and buttons, so one of the windowing environments is chosen, rather than writing our own.

2. Enhance Human-Computer Interaction

   Animated, dynamic feedbacks are commonly used in graphical environments to enhance the bidirectional interaction between users and the system. Feedback is continuously displayed before a complete command is issued, while it is impossible in a single command line based system.

   Another important feature in graphical environment is the use of icons to abstract concepts, operations and objects. Well-designed icons can be recognized and remembered more easily than textual description and may take less screen space [20].

3. Enhance Concurrency and Data Transfer

   One important feature pocessed by workstations of the latest generation is the multitasking capability. In a windowed, graphical environment, different programs run in separate, and probably overlapping windows on the same screen. Users can view those separate windows as separate terminals. Programs can use several windows to facilitate the display of different categories of data, and users can integrate the results from different windows. Multiple windows visualize the hardware concurrency while enabling human concurrency.

Following are some available graphical windowed environments: Microsoft Windows, OS/2 Presentation Manager, Macintosh Finder, X Windows, SunView, DecWindows, etc.

6.1.2. The Operating System

The ideal operating system for our system should support huge memory allocation with flat addressing[1]. Our system will support the presentation of pictures, which are bitmaps, and many other objects. There is a need for huge memory space. Moreover, for easy manipulation of bitmap, flat addressing is desired.

Popularity of the operating system is also very important. Potential users are not willing to switch to a new operating system where most of their daily applications cannot be executed. They will consider our system as another daily application, which should be running on the same operating system so that switching between applications will not bring any trouble.

6.1.3. The Hardware Requirement

If our system is to be popular, it must be executable on a popular and relatively cheap machine. The support for different peripherals should be plenty

1 Flat addressing means that a huge memory block has continuous locations from begin to end, despite of any segment limitations. The segment problem occurs in 80x86 based machines. Programmers need to tackle the segment limit problem by themselves.

because we require different devices like scanner, mouse, high resolution color display, removable hard disk, LCD projection panel, and even powerful RGB projector. IBM Personal Computer or its compatible is the obvious choice.

6.1.4. The Final Choice

If our system is to be executable on IBM personal computers or compatibles, only three windowing systems were needed to be considered: 1) Microsoft Windows, running on DOS; 2) OS/2 Presentation Manager, running on OS/2; and 3) X Windows, running on Unix. From the users viewpoint, applications running under these environments share the same look and feel. But obviously, DOS is the most popular operating system among the others in the personal computer world.

There are two operating systems that satisfy the huge memory requirement of the system: OS/2 and Unix. Virtual memory management is employed by both systems [2] [12] [25]. The size of a bitmap is limited only by the size of available memory (include virtual memory) in Unix. And OS/2 is expected to have the flat addressing feature in version 2.0 [12]. Both operating systems are not popular, but OS/2 has a novel feature of executing DOS programs in a compatibility box [23], making it a potential popular operating system. On the hardware requirement, Unix needs one hundred megabytes of hard disk space to hold both the base operating system and the X Windows environment, while OS/2 needs only ten megabytes. So OS/2 was chosen.

When comparing Microsoft Windows with OS/2 Presentation Manager, the difference is little, but significant. They share similar Application Programming

Interface (API). Programs for the two environments are very similar. The significant differences are that 1) OS/2 version 2.0 will support flat addressing; and 2) OS/2 is a preemptive multitasking environment while Microsoft Windows is not. For the first point, a discussion in section 6.1.2 is made. For the second point, programs for Microsoft Windows are required to be cooperative. When a program is executing an event handling routine, it takes over the CPU control. Other tasks will not be able to run before that task returns the control to Windows. If the event handling routine has a lot to do, it is the programmer's responsibility to break it down into smaller modules and coordinate the execution, which is tedious [18]. On the OS/2 side, true preemptive multitasking is employed. There is no need for the programmers to device methods to handle large jobs. From the developer's point of view, OS/2 Presentation Manager is a suitable platform for our system.

6.2. Representation of Objects

6.2.1. Basic Objects

Basic objects are represented by a STRUCT in C. As shown below:

```
struct GenericObj
{
   RECTL Boundary;
   COLOR Color;
   LONG MixMode;
   LONG LineWidth;
   LONG Pattern;
   BOOL Displayed;
   SHORT Layer;
   SHORT ObjectType;
   VOID *ObjDetail;
   SHORT ActiveSlot;
   struct GenericObj *Next;
   struct GenericObj *NextActive;
```

```
        struct GenericObj *Group;
        struct OperationList *Op;
    };
```

Each basic object is represented by a fixed part and a variable part. The fixed part consists of information such as color, width, pattern, display flag, and the bounding rectangle. The variable part contains information that is specific to different object classes, and is referenced by a pointer. For example, the variable part of the class *Text* contains the followings:

```
struct TextObj
{
   CHAR *Buf;
   SHORT TypefaceId;
   SHORT FontSizeId;
   USHORT Style;
   USHORT Height;
   USHORT Width;
};
```

Objects are linked together into a list. The list may contains grouped objects or objects of user-defined classes. They will be discussed in the next section.

6.2.2. Group and User-Defined Objects

When several objects are grouped together, they will be treated as one single object by the user. They will be moved and have color changed as a group. But the system will perform these operations on the objects one by one. So these objects are actually stored in the same format as before, except that they are grouped into a sub-list. A new object, of the type *Group,* which is used internally by the system, is inserted. An example of a *Group* object with three components is shown in figure 19.

Similar arrangement is made for user-defined objects, except that the node inserted into the list is given the user-defined class name rather than the *Group* type.

6.2.3. Set of Active Objects

The set of active objects are referenced by an array of pointers. The active objects themselves remain in the main object list so that little special treatment is needed to display them. After each refresh, the handles of the active objects are drawn, one by one. The use of array to reference the active object set will limit the total number of objects that can be active at a time. The current limitation is two hundred. Although seldom will the user selects more than this number, the active object set should be represented by a link list of pointers.

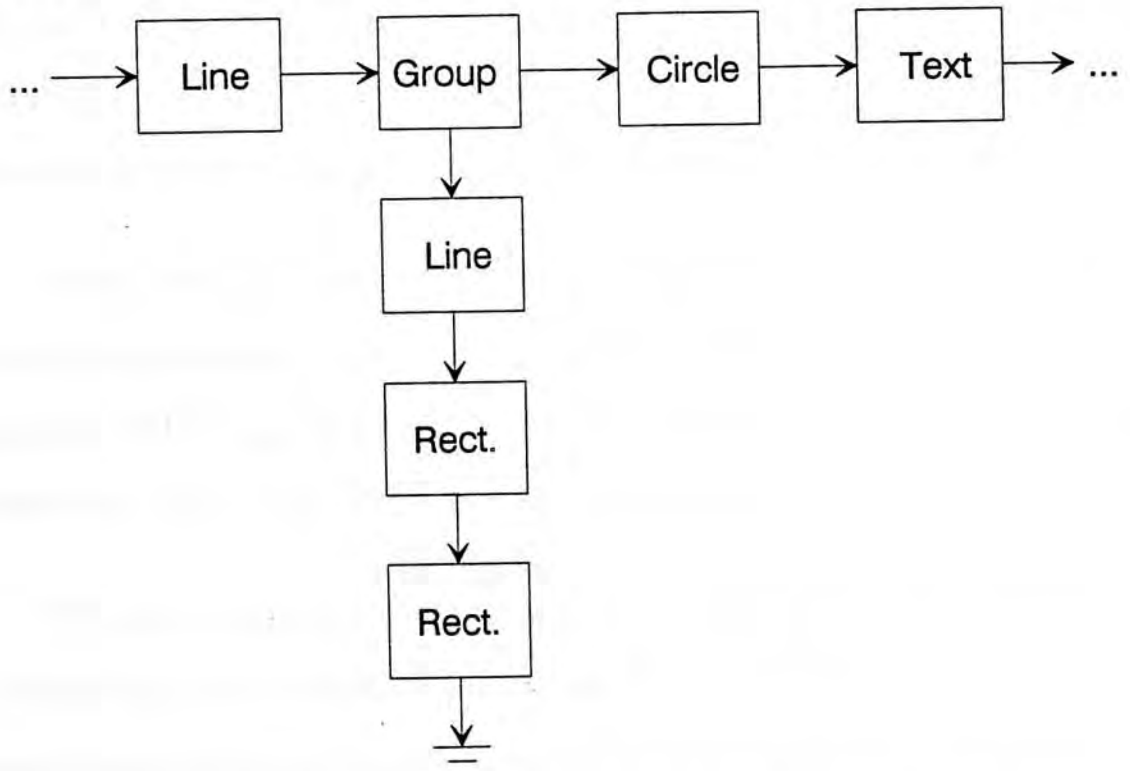


Figure 19. Internal representation of a *Group* object.

6.3. Object-Oriented Graphics Management Subsystem

Object-oriented graphics are employed in our system. Objects can be overlapped, resized and moved. When an object is moved or resized, some objects that are originally obscured will become visible, and some others that are originally visible may be obscured. An object-oriented graphics management subsystem is developed to handle the display and refresh of objects.

When an object is created, deleted, moved, resized, brought to front, or sent to back, the areas affected include the part which the object originally occupied and the part which the object currently occupying. The subsystem find out the union of these areas and inform the windowing system through a *WinInvalidateRect* API call. The windowing system will then call back the appropriate procedure, which is also a part of the graphics management subsystem, at the time when it is possible for refreshing.

At the time of refresh, the subsystem, which have access to all the physical attributes of all objects, will find out all the objects that have their bounding rectangles intersecting with the invalidated areas. And then these objects will be redrawn according to the stacking order. Those at the bottom will be drawn first.

The subsystem is powerful in managing and displaying the objects correctly. But sometimes extra redraw is performed. The subsystem compares the bounding rectangle of an object with the invalidated areas to check any intersections. Figure 20 shows several examples of 'false' intersections. This inefficiency might be solved by comparing not the bounding rectangles, but the drawing primitives with the invalidated areas.

The subsystem is relatively slow when used in animation, in which objects will be moved rapidly. A lot of invalidation, intersection checking and redrawing are needed. So another alternative should be used when animation is required at run-time. The object to be moved should be moved by 'BitBlt'[2] operation and mixed with the destination with XOR mode. Although the object being moved must be rectangular in shape, it will greatly improve the animation effect.

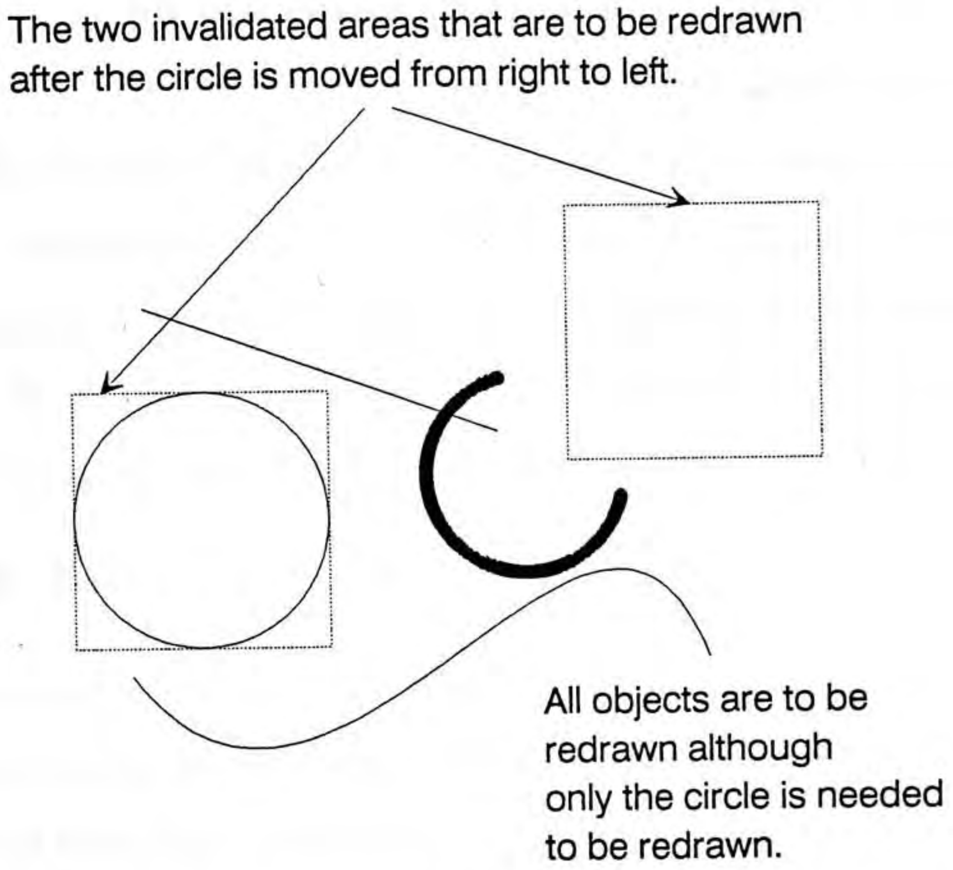

The two invalidated areas that are to be redrawn after the circle is moved from right to left.
All objects are to be redrawn although only the circle is needed to be redrawn.

Figure 20. 'False' intersections that lead to unnecessary redraw.

2 BitBlt refers to bit block movement. A bitmap is copied from a source area to a destination area [15].

6.4. Multiple Editing Window

In the current implementation, the user can open several overlapping editing windows simultaneously. This is achieved by maintaining a separate object list for each editing window.

6.5. Clipboard

Clipboard is a temporary area for the storage of objects that are to be transferred to and from other parts of the same application or even across applications. The clipboard is able to hold one object at a time. The object can be basic, grouped or user-defined objects. In the current implementation, only internal transfer of objects is possible through the clipboard. Nevertheless, it is very useful in transferring objects from one editing window to another.

6.6. Graphical Menu

Items in menu bars are usually textual. This is alright for operations such as loading or saving files. But for the selection of color, font, line width, pattern, etc., which can (and should) be visualized, graphical menus should be used. Our system uses actual colors, line width and pattern in the menus. Users are not required to translate the textual descriptions into their two dimensional equivalent. The selection of font and its size can also be implemented similarly. Figure 21 shows the line and pattern menu used in the system.

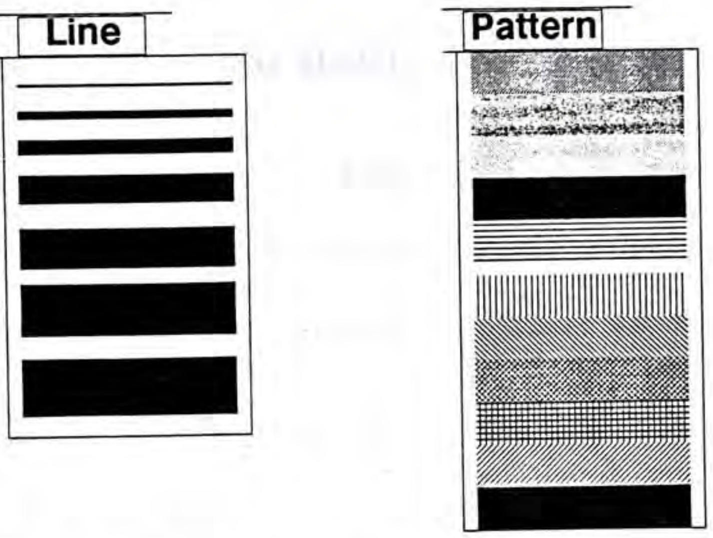


Figure 21. Graphical menu used in the system.

6.7. Font Management

In OS/2 Presentation Manager, fonts displayed on the screen are installable, that is, new fonts with different styles and supplied by various companies can be added at any time. Vector fonts are scalable. So virtually fonts of any sizes can be displayed. Ideally, our system should scan all the available fonts at run-time and let the user to use them all. Actual experiences found that the loading of a font and retrieving of the size information take a very long time. Scaling of font is also time-consuming, and the resulting visual effect is not good. Because of the above reasons, we use only five fonts which are shipped with OS/2: *System Proportional, Helvetica, Times Roman, Courier* and *System Monospaced.* And only limited sizes are allowed, while most of them are bit-by-bit mapping from memory to the screen, making no scaling required. The size of the available fonts are also saved to files. Each time when our system is started, the font information is read from the data files into the memory. This greatly enhances the execution speed. Lots of API function calls, which are the main reasons of slow execution, are avoided.

6.8. Mapping of the Active-Object-Set Model to the Implementation

The introduction of the Active-Object-Set Model not only helps non-programmers to have the idea of event processing, but also helps programmers to develop window-based event-driven programs.

The global views of the users and the programmers are basically the same. Both of them see a pool of objects, in which some of them are active. Operations are available to act on the objects and the identity of activeness can be transferred. The most important differences are that the operations available to the users and programmers are different, and different languages are used for the specification of these operations.

Operations available to the users are high-level ones, such as move, resize and change color. Other than these operations, the programmers face some low-level ones, such as draw the primitives, manipulate the object lists and active objects array, show and hide the mouse pointer, etc.

The implementation details of each type of basic objects are hidden from one another. They have standard interfaces to communicate with the graphics subsystem and input devices like the mouse and the keyboard. Adding a new object type is relatively easy. Only the drawing method and the event-handling routines are needed to be rewritten. The underlying framework has already taken control over the event redirection, message passing, notification of refresh, etc. Adding a new basic object is conceptually as easy as defining a new class from the user's point of view.

Adding a new operation is rather easy. No class specific object detail is needed if the operation is written for all kind of objects because the structure of the objects' common part is well defined. Common operations like move, reverse color, etc. can be written for existing classes and for those that might be added in the future.

The second major difference is the language of specifications. Users of the system specify objects and operations visually, while programmers specify them textually. The specifications from the user are interpreted each time, while the code written by the programmers are compiled. The job programmers do is tedious. In fact, if low-level operations can be available from the visual interface, what the programmers want to write might be specified visually, just like what the users do.

6.9. Representation of Operations

Basic operations are hard coded into the system. User-defined operations will be stored in memory and saved into data files when exit. When a new operation is defined, it will be represented by a list of sub-operations, as shown in figure 22. The type of objects that can be the parameter of the new operation will be stored in another list. Recall that the valid parameter types are selected in the dialogue box in figure 13. Assume the object that the new operation will act on is *A*. Accompanied with each sub-operation will be the object which the sub-operation will act on. This object can be the object *A* itself, or any sub-component of *A* if *A* is a user-defined object.

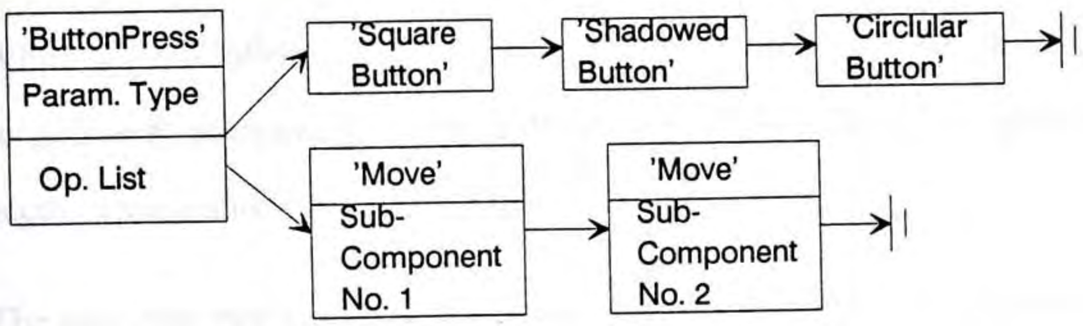


Figure 22. Representation of a user-defined operation.

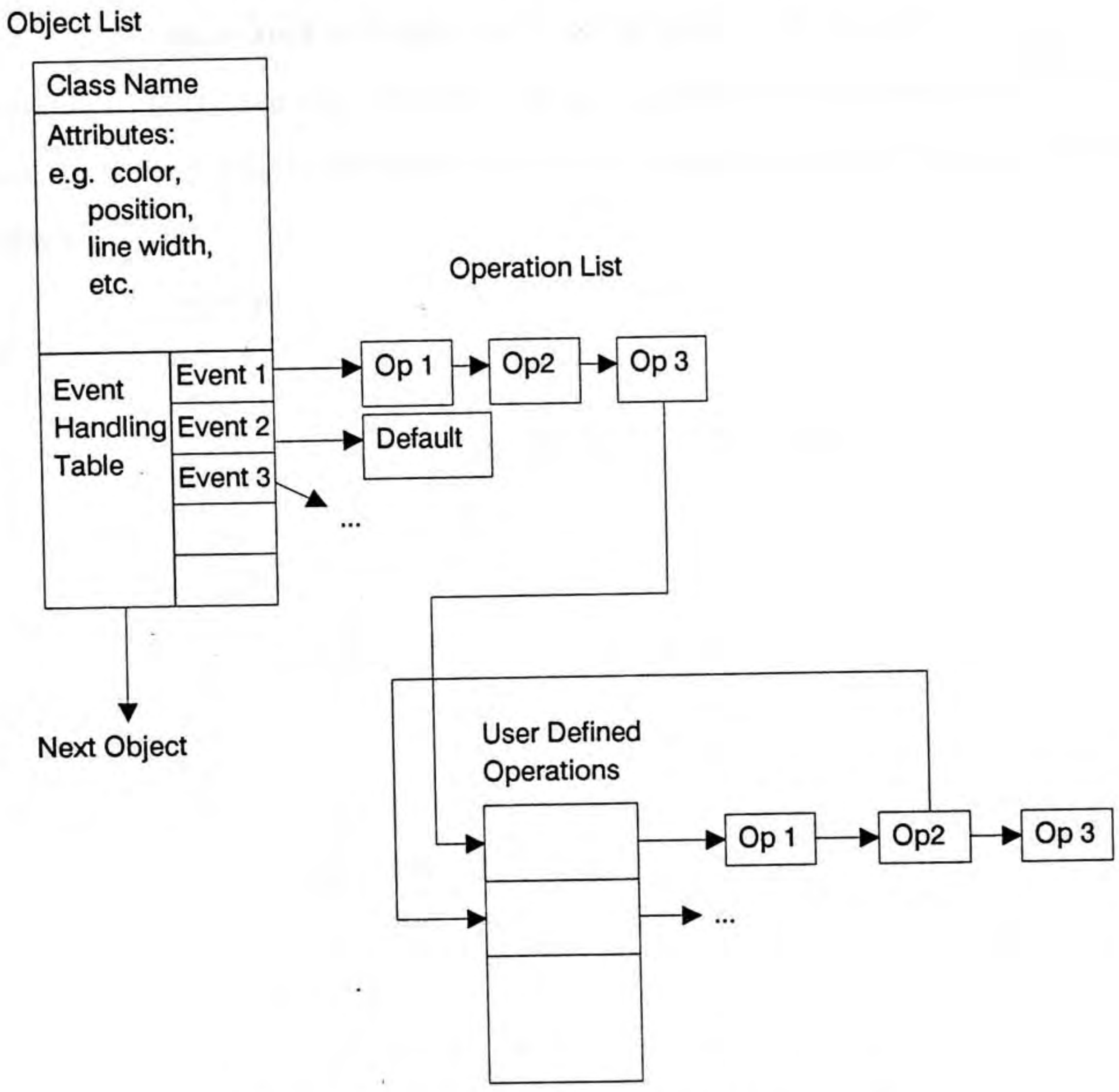


Figure 23. Operations that bound to an object.

When the operations are bound to the objects, each object in the object list will have pointers referencing the operations they should carry out when certain events occur. This is shown in figure 23.

The user may not bind any operation to an event happening on an object. If so, a default operation will be carried out when the event occurs. Default operations are associated with the classes, not an instance of a class. For each class of objects, and for each type of events that can happen on that class, there will be a default operation. Figure 24 shows the default operation table for different classes of objects.

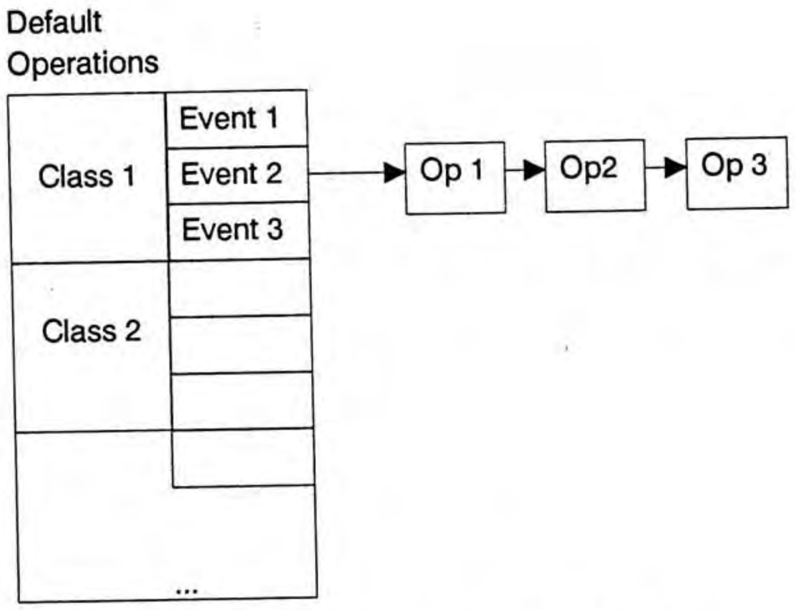


Figure 24. Default operations table.

Chapter 7
Future Work and Conclusions

This chapter will discuss the limitations of the current design, how the system can be improved, and the areas that our design can be extended to. And then a conclusion of the project will be given.

7.1. Limitations

7.1.1. Direct Manipulations

Our system employs direct manipulations in most situations. From creation and modification of objects to definition of new operations. But certainly there are still situations that the interactions can be changed into direct manipulations. For example, the cut-and-paste operation using the clipboard can be done more naturally by allowing the user to drag object from and to the clipboard. The clipboard can be visualized as a window. Dragging objects from current editing window means cutting objects to the clipboard. Dragging objects from clipboard means pasting objects from it. Another example is the *Clear* operation which delete objects from the current editing window. A more natural approach, employed by the Macintosh interface, is to provide a *Trash* icon which allows users to dump useless objects into it.

7.1.2. Multiple Presentation Windows

The current system provides multiple editing windows to let the user transfer material from one document to the others. But during the presentation, only one presentation window is provided. Multiple and properly overlapping presentation windows are desired so that more information can be conveyed at the same time, and important materials can be retained in one window while other materials can be continued in another window.

The number of presentation windows provided should not be limited, although seldom will a presentation need more than a dozen. The number of presentation windows needed depends mainly on the nature of the presentation, and is usually limited by the size of the screen.

7.1.3. Editing of User-Defined Operations

In the current design, the user demonstrates the new operation while the system records the sequence. The user has no means to get a list of operations recorded, nor can he/she further edit the sequence of operations. The only way to modify a user-defined operation is to re-record them.

If the user-defined operation is very complex, or is recorded by another user, then it may be impossible to re-demonstrate the sequence. The system should provide a tool for the visualization of operation sequences. This tool can be used to monitor the execution of a user-defined operation also. The function of this tool will be similar to a debugger which is used in traditional programming.

7.2. Future Work

7.2.1. Maintaining Relationship Through Constraint Satisfaction

A constraint specifies a relation that must be satisfied at all times between two objects. In a graphical application, an object not only can affect/be affected by the others, its behaviour may be restricted by the others. The relationships to be maintained include object layout, dynamic feedback, etc., which involve the static and dynamic attributes of the objects. For example, a user-defined object class, *Button*, may involve two components, the button text and the box containing the text object. It is desired that the text box should always be larger than the text object for a fixed offset, and the text object should be always located at the center of the box. This is an example of constraints that concerns with the static attributes of objects. Another example concerning the dynamic attribute of objects is the definition of the *ReverseColor* operation. This operation can be defined as putting a black rectangle on top of the object to be reversed, with XOR mode. The constraint involved is that the black rectangle put on top must be as large as the object to be reversed.

Users should be able to specify the constraints visually. Again, all we concern about are visible objects which have position, dimensions, etc. as attributes. Normally, these attributes are quantities that are measurable (e.g. height and width), or enumerable (e.g. color), enabling the constraints to be represented by mathematical equations [5]. Current drawing programs allow users to specify the alignment constraints, such as, centering of a group of objects along a certain place marker; restricting a line to be horizontal, vertical or inclined at an angle of 45 degrees. These constraint information is not saved. They are only used during the construction of objects. Moreover, many other useful constraints such as parallel lines, equal length, same color, object containment, etc. have no well developed visual interaction techniques to specify.

Another issue is to satisfy the constraints at run-time. An incremental constraint-solving algorithm would be required to identify which constraints must be reevaluated and limit its solutions to meet with these constraints [29]. When a certain event that modifies the active object occurs, the active object will first find out the desired new value for each attribute, then a verification process done by related objects will be invoked. If the verification failed, that is, a constraint is violated, a suggestion from the verification routine should be submitted.

7.2.2. Functions for System Status/Values Query

A group of functions should be provided to help the user to refer to objects such as the active set, the next selected object[1], the clipboard object, the mouse pointer location, etc. Similarly, functions like those returning the attributes of the active set are necessary, so that an operation can depend on the attributes of other objects.

7.2.3. Private Status Flag, Pre-Conditions and Conditional Execution

A complete interaction sequence may involve more than one event, in a predefined order. Because the binding of operations and events does not involve the ordering information, some status flags must be set to remember the state of the interaction sequence.

In our design, an event sent to an object would invoke a sequence of operations. This would limit the event handling capabilities of an object. The following example cannot be done using the current design: Suppose we want to create an icon that, on the first click, it will activate a pop-up box containing some

1. The next selected object is defined to be the object currently pointed to by the cursor. This object may not necessarily be a member of the active set.

information. On the second click, it will dismiss the pop-up box. If another click is generated again, the sequence repeats. This kind of interactions cannot be done by our design because the mouse click event is allowed to be bound to the same sequence of operations only. What the user can do in the current design is to activate the pop-up box when the icon is being pressed, and dismissed the pop-up when the mouse button is released. The pop-up activation and dismissal are associated with two events, *MousePress* and *MouseRelease,* respectively.

The use of private status flag can solve this problem. A flag can be created for remembering the status of the pop-up box, whether it is currently visible or not. And different operations can be carried out depending on the value of the status.

Private status flags are useful in controlling the interaction sequence also. Many interaction paradigms consist of several interaction events, which must be occurred in sequence. One example is the dragging of an object. This interaction consists of the following sequence of events: *MousePress, MouseMove* and *MouseRelease.* Although *MousePress* must be occurred sooner than *MouseRelease,* and no explicit checking is needed, *MouseMove* must be guaranteed to be occurred after the *MousePress* and before the *MouseRelease.* The introduction of private status flag should be able to solve this problem.

It is possible to restrict the operation, in response to the occurrence of an event, to be carried out only if certain pre-conditions are satisfied. Pre-conditions are used to ensure the correct interaction sequence. Moreover, it can enable the conditional execution of different actions. Event may be associated with not only a single operation, but with several pre-conditions/operations pairs. When an event occurs, each pair of preconditions/operations are checked, the associated operation of the first pre-condition satisfied would be carried out.

Figure 25 shows the relations between objects, events and operations. Figure 26 gives a sample sequence of the activation of an operation.

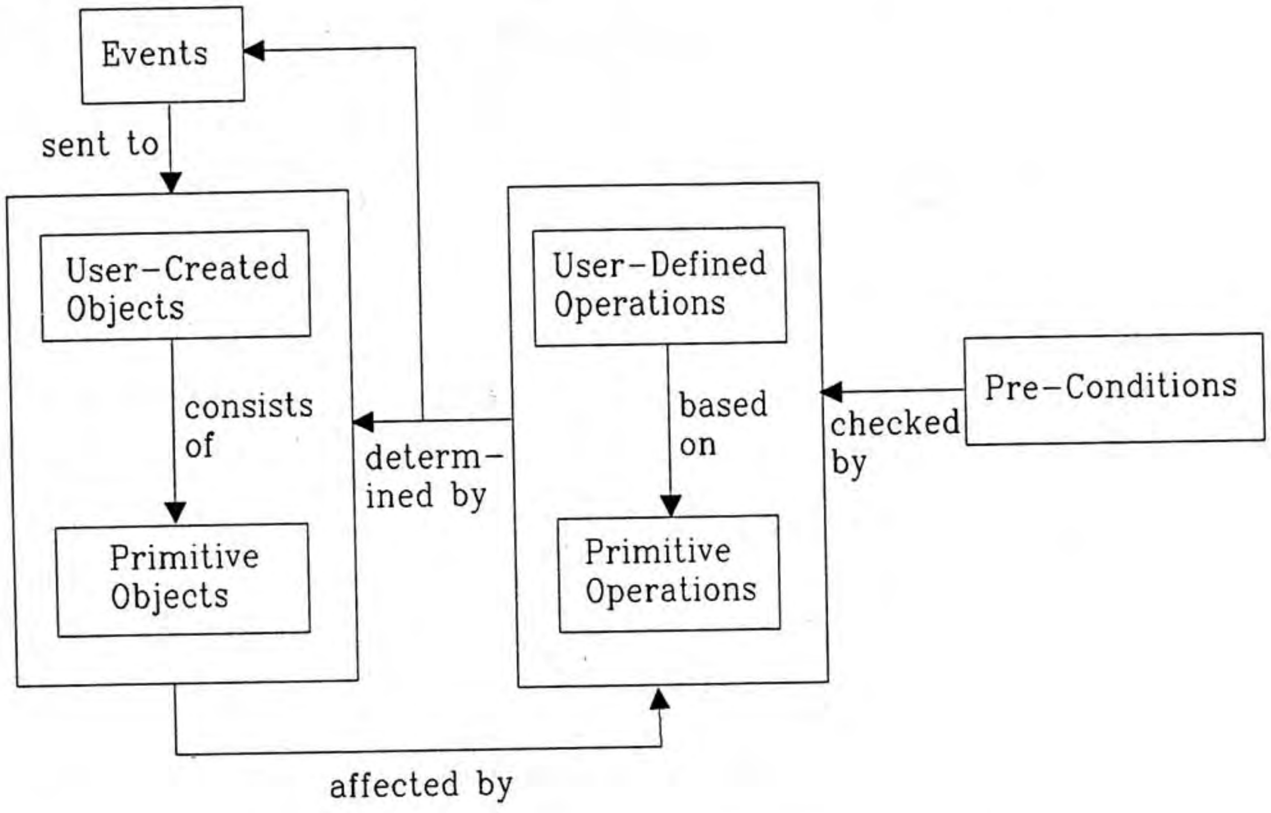


Figure 25. Relations between object, event and operations.

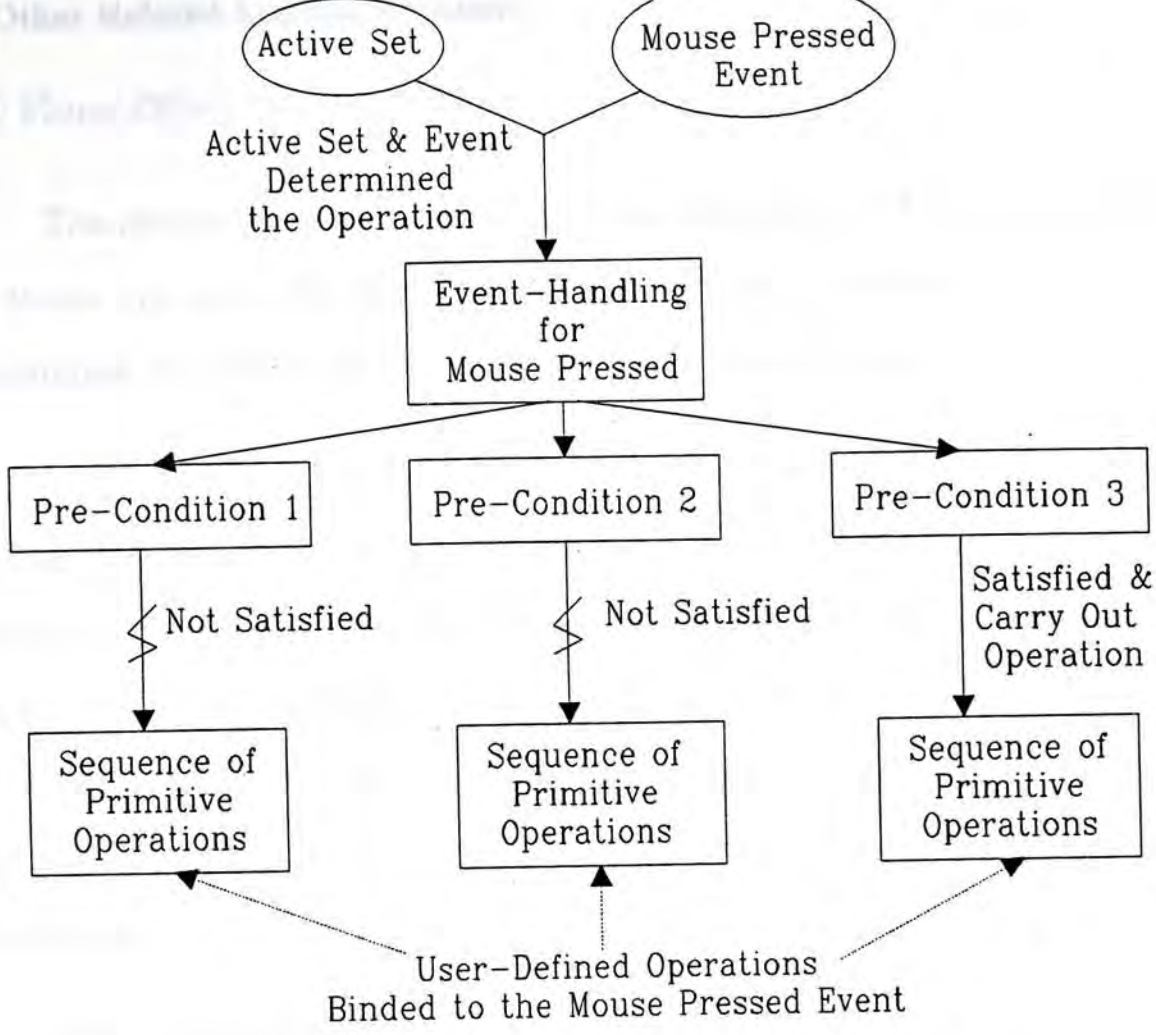

Active Set
Mouse Pressed Event
Active Set & Event Determined the Operation
Event-Handling for Mouse Pressed
Pre-Condition 1
Pre-Condition 2
Pre-Condition 3
Not Satisfied
Not Satisfied
Satisfied & Carry Out Operation
Sequence of Primitive Operations
Sequence of Primitive Operations
Sequence of Primitive Operations
User-Defined Operations Binded to the Mouse Pressed Event

Figure 26. Execution sequence for an event.

7.2.4. Object Oriented Programming

Although the current design is based on graphics objects, the implementation itself does not employ any object oriented programming techniques. No class inheritance is concerned. Future enhancement of the system should be migrated to an object oriented language such as C++ and Smalltalk.

7.3. Other Related Application Areas

7.3.1. Visual-Object Oriented Systems

The Active-Object-Set Model and the interaction techniques introduced in this thesis are not only applicable to the area of courseware production and presentation, but also to other *Visual-Object Oriented Systems.*

Visual-Object Oriented Systems refer to those systems that the main purposes are to create, display, modify and manipulate visible objects. Visible objects refer to anything displayable on the screen, including text, geometric line and curve, pattern, bitmap, diagram, and even free-hand drawing. Examples of Visual-Object Oriented Systems are word-processor, drawing program, CAD program, and User Interface Management System, which will be elaborated in the next section.

The Active-Object-Set model can be applied to Visual-Object Oriented Systems in general. But the basic objects, basic operations and events concerned differ from one system to another. For example, the object classes that a word-processor is concerned may include *Letter, Word, Paragraph,* and *Document,* while each is a sub-class of the following one. Other object classes include *Cursor, MarginLine, Header, Footer,* etc. The operations include *Scroll, ChangeLineSpacing, Justification,* etc. Clearly, not all operations can be defined visually in an easy way. And the basic operations provided may be at a lower level than what are provided in the Courseware Production and Presentation System.

One should not expect a complete visual-object oriented application to be defined completely using visual interaction techniques[2], but at least many visual object manipulation routines, which are quite troublesome for the developers, should be able to define without textual specification.

In the field of visual programming, a three-dimensional framework is proposed to access visual programming languages in a qualitative way [35]. The language level, the scope of applicability and the visual extent are used to measure a visual programming system. For example, Xerox's Star system [13], the classical iconic system, has a very high visual extent, but low in both language level and scope of applicability (figure 27). Our system will have a similar profile for which all the definitions of objects and operations can be performed visually, but the scope of applicability is small. The language level supported is even lower because we do not support explicitly any programming constructs.

2 The spell check utility found in most word-processor is difficult to define visually, although this part of the application cannot be viewed as a visual-object oriented component.

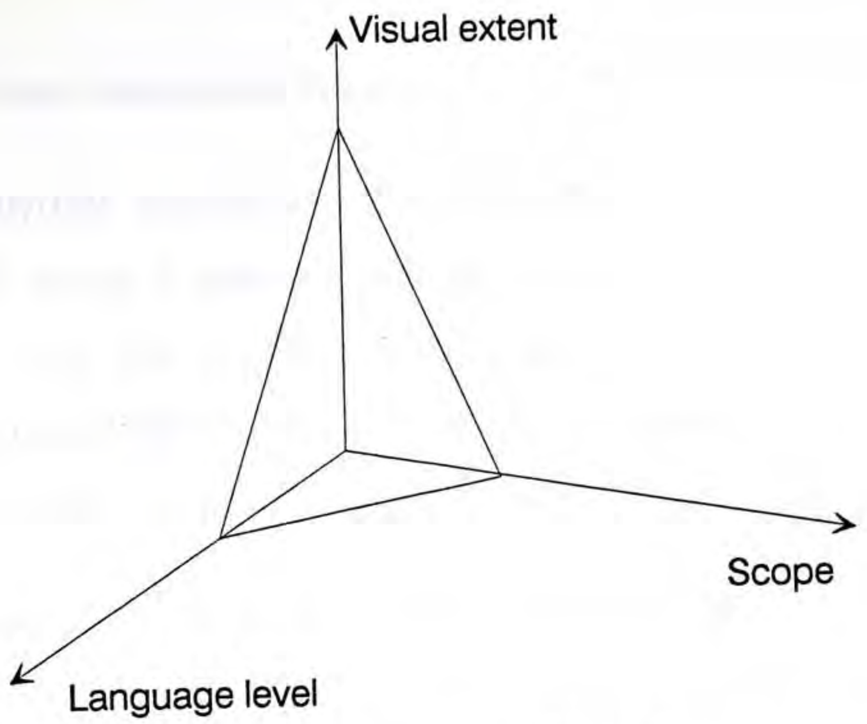


Figure 27. The profile of Xerox Star.

If similar approach is used in the development of visual-object oriented applications, a similar profile would be obtained. Certainly, a complete application must be developed with a tool that is of high language level. Only some of the aspects like visual interactions and visual objects manipulations can be programmed by visual means. Other parts, such as file management and spell check would properly be developed using traditional programming languages. There must be ways to integrate the results of the two parts. A viable approach is to generate source code files for those program components that are defined visually.

A more mature application that current researches are trying to apply visual interactions on it is the User Interface Management System, which is discussed below.

7.3.2. User Interface Management Systems

User Interface Management System (UIMS) is, in fact, a kind of visual-object oriented system. It manages both the static (appearance of menu, button, dialogue box, etc) and dynamic (interaction sequence, dynamic feedback) attributes and behaviours of the user interface. A current research [29] suggests that the user interface component can be defined visually.

If our design can be applied to the area of UIMS, user interface designer, which properly are non-programmers, can put all their effort in the design work. Rapid prototyping can be achieved.

7.4. Conclusions

In this thesis, the design and implementation of the Courseware Production and Presentation System are discussed. Several research areas are of interests, namely, Authoring System, Visual Programming and User Interface Management System.

The authoring system, together with a few commercially available presentation softwares are useful in providing hints for what kind of presentation features should be included in our system. The graphical interfaces commonly used by these system prompt a clear environment for the final product to be executed in. And OS/2, with the Presentation Manager graphical user interface, is selected to be the environment after careful comparisons with other systems like Microsoft Windows and X Window on Unix.

The Peridot system [28] and the Programming by Rehearsal system [16] are the main stimulations to the development of the Active-Object-Set model. Every object in the system, which composes of graphics primitives, will response to external events and carry out certain operations, acting on itself or on others. And the static and dynamic attributes and behaviours can be defined through visual interaction techniques. This can be viewed as a simple visual programming system, which has limited scope, but a great visual extent.

The concept of defining object properties visually can be extended to other visual-object oriented applications, including the field of User Interface Management Systems.

The system implemented is only a coarse prototype of the final one. It just demonstrates the possibility of the suggested approach. There is still a long way from a complete, robust commercial product. And there would be more problems encountered (e.g. the execution speed) as the project continues.

References

1. Apple Computer Inc. *HyperCard User Guide.*

2. Bach, M. J.*The Design of UNIX Operating System.*Prentice Hall, New Jersey, 1986.

3. Balagopalan, Santosh and MacKnight, Carol B. Authoring Systems: Some Instructional Implications. *J. Educational Technology Systems.* 17, 2, 1988-89, p123-134.

4. Berry, R. E. Common User Access - A Consistent and Usable Human-Computer Interface for the SAA Environments. *IBM Systems Journal.* 27, 3, 1988, p281-300.

5. Borning, A. The Programming Languages Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory. *ACM Transactions on Programming Language and Systems.* 3, 4, Oct. 1981, p353-387.

6. Brad, J. C. *Object-Oriented Programming, An Evolutionary Approach.* Addison-Wesley, 1986.

7. Brown, G. P., Carling, R. T., Herot, C. F., Kramlich, D. A., and Souza, P. Program Visualization: Graphical Support for Software Development. *IEEE Computer.* Aug. 1985, p27-35.

8. Calder, P. R., Linton, M. A., and Vlissides, J. M. Composing User Interfaces with InterViews. *IEEE Computer.* Feb. 1989, p8-22.

9. Cardelli, L., and Pike, R. Squeak: A Language for Communicating with Mice. *Proceedings of ACM SIGGRAPH '85.* Jul. 1985, p199-204.

10. Chang, S. K., and Kunii, T. L. Pictorial Database Systems. *IEEE Computer.* Nov. 1981, p13-21.

11. Chang, S. K. Visual Language: A Tutorial and Survey. *IEEE Software.* Jan. 1987, p29-39.

12. Duncan R. OS/2 Version 2.0: Exploiting the 32-bit Architecture of 80386- and 80486-based Systems. *Microsoft Systems Journal.* May 1990, p1-14.

13. Farrel, J., Klose, P., and Purvy, R. The Design of Star's Records Processing: Data Processing for the Non-computer Professional. *ACM Transactions on Office Information Systems.* Jan. 1983. p3-24.

14. Feiner, Steven. An Experimental System for Creating and Presenting Interactive Graphical Documents. *ACM Transactions on Graphics.* 1, 1, Jan. 1982, p59-77.

15. Feiner, S. K., Foley, J. D., Hughes, J. F., and van Dam, A. *Computer Graphics: Principles and Practice (2nd ed.).* Addison-Wesley, 1990.

16. Finzer, W. and Gould, L. Programming by Rehearsal. *BYTE.* Jun. 1984, p187-210.

17. Glinet, E. P., and Tanimoto S. L. Pict: An Interactive Graphical Programming Environment. *IEEE Computer.* Nov. 1984, p7-25.

18. Hall, W. S. Adapting Extended Processes to the Cooperative Multitasking of Microsoft Windows. *Microsoft Systems Journal.* Jan. 1991, p21-34.

19. Hirakawa, M., Monden, N., Yoshimoto, I., Tanaka, M., and Ichikawa, T. HI-VISUAL: A Language Supporting Visual Interaction in Programming. *Visual Languages,* ed. by S. K. Chang et. al. Plenum Press, 1986, p233-259.

20. Huang, K. T. Visual Interface Design Systems. *Principles of Visual Programming Systems.* Prentice Hall. 1990.

21. Jonassen, David H. *Hypertext/Hypermedia.* Educational Technology Publications, Inc., New Jersey, 1989.

22. Kearsley, Greg. Authoring Systems in Computer Based Education. *Communications of the ACM.* Jul. 1982, p429-437.

23. Kogan, M. S., and Rawson, F. L. The Design of Operating System/2. *IBM Systems Journal.* 27, 2, 1988, p90-104.

24. Kuo, I., and Wong, H. K. T. GUIDE: A Graphical User Interface for Database Exploration. *Proceedings of the Conference on Very Large Databases.* 1982, p22-32.

25. Letwin, G. *Inside OS/2.* Microsoft Press, Washington, 1988.

26. Meskil, Carla. Interactivity in CALL Courseware Design. *CALICO Journal.* Sep. 1987, p9-14.

27. Myers, B. A. INCENSE: A System for Displaying Data Structures. *ACM Computer Graphics.* Jul. 83, p115-125.

28. Myers, B. A., *Creating User Interfaces by Demonstration.* Technical Report CSRI-196, University of Toronto, May 1987.

29. Myers, B. A., Giuse, D. A., Dannenberg, R. B., Zanden B. V., Kosbie D. S., Pervin, E., Mickish, A., and Marchal P. Garnet: Comprehensive Support for Graphical, Highly Interactive User Interfaces. *IEEE Computer.* Nov. 1990. p71-85.

30. Olsen, D. R. Larger Issues in User Interface Management. *ACM Computer Graphics.* Apr. 1987, p134-137.

31. Petzold, C. *Programming the OS/2 Presentation Manager.* Microsoft Press, Washington, 1989.

32. Pfaff, Gunther R., ed. *User Interface Management Systems.* Springer-Verlag, 1985.

33. Raeder, G. A. A Survey of Current Graphical Programming Techniques. *IEEE Computer.* Aug. 1985, p11-25.

34. Schmucker, K. J. MACAPP: An Application Framework. *BYTE.* Aug. 1986, p72-75.

35. Shu, Nan C. Visual Programming Languages, A Perspective and a Dimensional Analysis. *Visual Languages,* ed. by S. K. Chang et. al. Plenum Press, 1986, p11-34.

36. Shu, Nan C. *Visual Programming.* Van Nostrand Reinhold, New York, 1988.

37. Young, D. A. *X Window Systems: Programming and Applications with Xt.* Prentice Hall, New Jersey, 1989.