An Object-Oriented Methodology for Modern User Interface Development

by

LAM Siu Hong

A THESIS

Submitted to
The Chinese University of Hong Kong
in partial fulfillment of the requirements
for the degree of

MASTER OF PHILOSOPHY

Department of Computer Science

May 1991

325481
thesis
QA
76.9
U83 L35

Acknowledgments

I would like to express my deep gratitude to my supervisor Dr. Moon and other M.Phil students in Computer Science Department for giving me valuable advice and helpful criticism.

Abstract

In order to increase the usability of computer system, user interfaces become more and more sophisticated and complicated. Some new user interface features such as separating user interface from application, direct manipulation, undo / recovery and multi-thread dialogue are introduced into modern user interfaces. These features are very attractive and interesting but they also cause many technical problems during the modern user interface development life cycle. These technical problems include concurrency of dialogue control, multiple continuous feedbacks, recovery of user interfaces at different abstract levels ...etc. Due to these technical problems, the software development and maintenance cost of modern user interfaces keep increasing in the past few years.

Focusing on the above problems, this thesis proposes an Object-Oriented methodology for developing modern user interfaces so that much of the above mentioned difficulties can be solved. Ideally, with this Object-Oriented methodology, the development time of user interfaces can be reduced and the development processes of user interfaces can be systematic and structured. Modification of user interface becomes easy and efficient. An Object-Oriented User Interface Model and a User Interface Framework whose design is based on this model are proposed in this thesis so as to achieve the above objectives.

An Object-Oriented Methodology for Modern User Interface Development

Chapter 1
Introduction 1
1.1 Software Development Crisis of User Interface 1
1.2 Objectives and Scope of Interests 1
1.3 Overview of the Thesis 2

Chapter 2
Background and Problems 4
2.1 Categories of User Interfaces 4
2.2 Trends of User Interfaces 6
2.3 Some other Desirable Features and Problems of UI Development 7
2.3.1 Separating UI from Application 7
2.3.1.1 Benefits of Separable UIs and Applications 7
2.3.1.2 Requirements of Complete Separation 10
2.3.2 Instant Continuous Feedback 12
2.3.2.1 Problems of Linguistic Model on World Model Type UIs 12
2.3.3 Undo and Recovery 15
2.3.4 Iterative Design through Rapid Protyping 16

Chapter 3
An Object-Oriented Model for Model World User Interfaces Development 18
3.1 Features of UIs to be supported by the Model 18
3.2 A Linkage Model for Separating UI from Application 19
3.2.1 Communication Messages Modeled using an Object Oriented Approach 20
3.2.2 A Sample Message 22
3.2.3 Linkage in a Distributed Heterogenous Environment 24
3.2.4 Comparing the Linkage Model with the Application Interface Model in Seeheim's UI Model 25
3.3 An Object-Oriented Model for Supporting Multiple Feedbacks and Multi-thread dialogue 26
3.3.1 An Overview of the Model 27
3.3.2 Objects on the Lexical Layer 28
3.3.3 Roles of Presentation Objects 29
3.3.4 Syntactic Objects 31
3.3.5 Interaction Objects 32
3.3.6 Interaction between objetcs and Linkage Component 33
3.3.7 Multiple U-tubes Ladder for Supporting Multiple Feedbacks 33
3.3.8 Recovery through a Generic UNDO stack 35
3.3.9 Dialogue Control in an Object 37
3.3.10 Interactive Objects 39

3.3.11 An Architecture for Supporting Multi-thread Dialogue 40
3.4 Basic Object Structure 42
3.4.1 An Event Model for Dialogue Control 43
3.4.2 Maintain Consistency through ϵ-rules 45
3.4.3 An Example of an Inner Object Specification 47
3.4.4 Pre and Post Condition of Action 49
3.4.5 Automatic Message Routing 49
3.5 Systematic Approach to UI Specification 50

Chapter 4
User Interface Framework Design 52
4.1 A Framework for UI Development 52
4.1.1 Abstract Base Class for Each Object Type 54
4.1.2 A Kernel for Message Routing 60
4.1.3 Interaction Knowledge Base 63
4.1.4 A Dynamic View of UI Objects 64
4.1.5 Switch Box Mechanism for Dialogue Switching 66
4.1.6 Software IC Construction 68
4.2 Summaries of Object-Object UI Model and UI Framework 70
4.2.1 A New Approach to User Interface Development 70
4.2.2 Feautures of UI Development provided by the Object-Object UI Model and UI Framework 71

Chapter 5
Implementation 73
5.1 Implementation of Framework in MicroSoft Window Environment 73
5.1.1 Implementation of automatic message routing through dynamic binding 73
5.1.2 A generic message structure 75
5.1.3 A meta class for object communication 76
5.1.4 Software component of UI framework in Microsoft Window environment 76
5.2 A Simple Stock Market Decision Support System (SSMDSS) 77
5.2.1 UI Specification 81
5.2.2 UI features supported by SSMDSS 87

Chapter 6
Results 89
6.1 Facts discovered 89
6.1.1 Asynchronous and synchronous communication among objects 89
6.1.2 Flexibility of the C++ language 90
6.2 Technical Problems Encountered 91
6.2.1 Problems from Implementation Platform 91
6.2.2 Problems due to Object Decomposition in an Interactive Object in SSMDSS 92

6.3 Objectives accomplished by the Object-Oriented UI Model indicated by SSMDSS 93

Chapter 7
Conclusion 95
7.1 Thesis Summary 95
7.2 Merits and Demerit of the Object-Oriented UI Model 96
7.3 Cost of the Object-Oriented UI Model 96
7.4 Future work 97

Appendix
A1 An Alogrithm for Converting Transition Network Diagram to Event Response Language A1
A2 An Object-Oriented Software Development A4
A2.1 Traditional Non Object-Oriented Software Development A4
A2.2 An Object-Oriented Software Development A6
A3 Vienna Development Method (VDM) A8
A3.1 An Overview of VDM A8
A3.2 Apply VDM to Object-Oriented UI model A10
A4 Glossaries and Terms A12

Reference

Chapter 1

Introduction

This chapter presents the basic motivations and objectives of our research. An overview of this thesis is also presented in this chapter.

1.1 Software Development Crisis of User Interface

User Interface (UI) is a crucial factor determining computer system usability. A good UI can encourage a user to make full use of a computer application and hence the user can increase his/her productivity or efficiency in using a computer. Although a good user interface is desirable, it is not easy to develop. Besides such human factors as psychology or culture that make the specification of UIs difficult to write, many new UI features such as undo, direct manipulation, and multi-thread dialogue, make modern UIs very sophisticated and complicated. These new UI features also introduce many technical problems, such as concurrency of dialogue control, multiple continuous feedbacks, recovery of UI at different abstract levels ... etc., to modern UI development. In order to handle the above technical problems, codes of modern UIs are usually bulky, unstructured and difficult to maintain. Consequently, UI development usually occupies major part of a software product's development time and contributes a bottle neck in software development cycle.

1.2 Objectives and Scope of Interests

The objective of this thesis is to propose a new approach to modern UI

development so that UI development time can be greatly reduced. Moreover, the development of UI becomes more systematic and structured. According to this new UI development approach, we can make UI maintenance work easy and efficient. In short, with the help of this new UI development approach, the software development and maintenance cost of UI can be reduced. An Object-Oriented UI Model is proposed in this thesis so as to achieve the above objectives. In the following chapters, we will show how the object-oriented paradigm can provide us a comfortable and efficient environment for UI development.

Although UI development also requires knowledge from different domains such as psychology or ergonomics to construct a user model for UI requirement specification [1,15,61,63], we assume that we can get around this by iterative designs through rapid prototyping as described in section 2.3.4. Hence, in this thesis, we restrict our research interests only to the technical problems caused by the new UI features described above.

1.3 Overview of the Thesis

Chapter 2 presents relevant background materials such as categories of UIs and the trends of UIs. Some desirable features of modern UIs are identified. Chapter 2 also presents the development problems of modern UIs.

An Object-Oriented UI model is proposed in chapter 3. How this model can give solutions to such problems as separating UI from application, multiple continuous feedbacks, undo/recovery ... etc, will be discussed. Chapter 3 also introduces a new approach to specify the individual software components and

their dialogue controls in a UI.

Chapter 4 describes the design of a UI framework which is a basic blue print for our UIs development. The design of this UI framework is based on the Object-Oriented UI Model described in chapter 3. The objective of UI framework is to provide an easy and comfortable environment for UI development. Rapid evolutionary prototyping of UI is also supported in this UI frame environment.

Chapter 5 presents the implementation of the UI framework in the MicroSoft Windows environment. A Simple Stock Market Decision Support System (SSMDSS), which is implemented according to the UI framework, is presented. The implementation of SSMDSS is used as an example to illustrate the properties of the Object-Oriented UI Model and UI framework.

Chapter 6 presents the results of SSMDSS. The difficulties and problems of implementing this system are described. The accomplishments of the Object-Oriented UI model indicated by this implementation of SSMDSS are identified.

Chapter 7 summarizes this thesis. It also points out the merits and demerits of the Object-Oriented UI model. Finally, future works are suggested.

Chapter 2

Background and Problems

In this chapter, we identify the trends of modern user interfaces. The needs and justifications of some desirable features of modern user interfaces are discussed. However, these desirable features also introduce many new problems and challenges to user interface designers. Finding a new user interface development approach to cope with these problems is the primary goal of our research.

2.1 Categories of User Interfaces (UIs)

Based on the interaction styles of UI, we can classify UIs into two main streams.

```
C:\>
C:\>dir temp

 Volume in drive C is S H LAM
 Directory of C:\

TEMP                              51   11-07-88    2:22a
              1 File(s)      2764800 bytes free
C:\>
C:\>del temp
```

Figure 2.1 An example of conversational world style UI.

Conversational world style: An example of conversational world style UI is shown in Figure 2.1. UIs that fall into this stream treat human-computer interaction as human conversation in which each participant speaks in turn. A user inputs a command line into a computer and the computer, according to the

grammar of the command language, interprets the meaning of the input command and then acts upon the input command and produces some outputs to the user. The user in turn interprets the output of the computer and gives another input command. The cycle repeats until some goals are accomplished. The dialogue between the user and the computer is sequential and is supposed to move in a predictable manner which can be described by a finite automata. This style of interaction is adopted by many conventional text-based interfaces and has been well modeled by many linguistic models such as language parser and argument transition network [29,32].

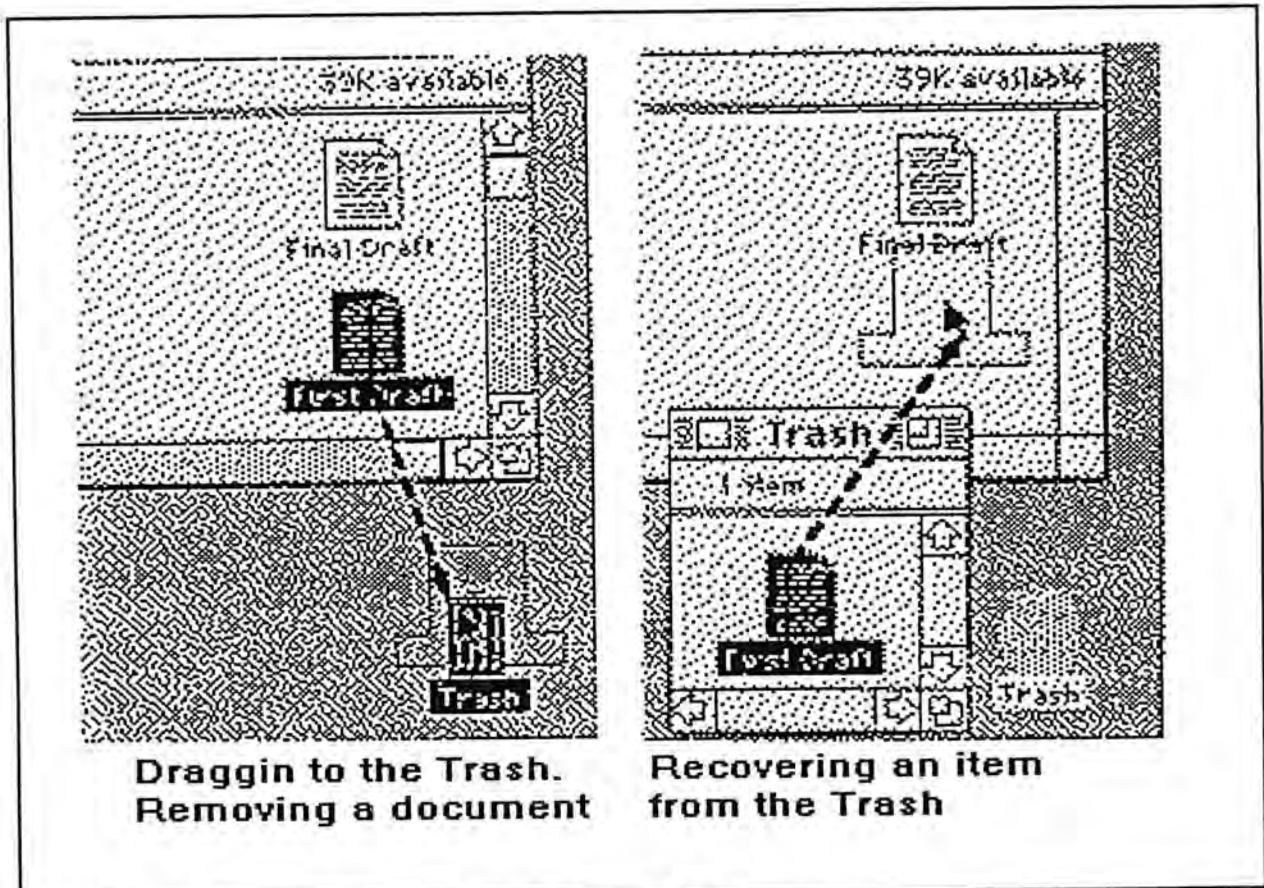


Figure 2.2 An example of Model world UI.

Model world style [32,33]: An example of model world style UI is shown in Figure 2.2. This style of interaction tries to represent real world objects visually, such as radio button and file's icon, so that a user can manipulate the objects directly through some input devices such as mouses or light pens. Unlike

conversational world style UIs, objects being manipulated are directly presented to a user rather than remaining abstract. Model world style interaction also provides instant continuous feedback to a user. For example, a file's icon moves continuously when the icon is being dragged by a mouse.

Another important feature of model world style interaction is multi-thread dialogue. This feature makes this style of interaction appear modeless. In essence, a user may suspend a dialogue with an object at one time and switches to another object to start a new dialogue with that new object immediately. After the user has finished the dialogue with the new object, he or she can switch back to the original object and resumes the dialogue from the point it is suspended. Because of this multi-thread dialogue interaction, users can have dialogues with several independent objects at one time. The dialogue between user and computer is asynchronous as a user can switch to other object or task whenever he or she wishes. Multi-windowing system is a good example of multi-thread dialogue interaction.

Such style of interaction is adopted by most graphical UIs and windowing systems such as the UI builder in NeXT and the MacApp in Macintosh [72].

2.2 Trends of UIs

With the advent of modern low-cost graphics hardware and popularity of personal computers for laymen, more and more UIs are designed in the direction of model world style interaction. As this style of interaction can simulate our real world objects in graphical forms (e.g. the desktop environment in Macintosh), this interaction is close to our daily life and hence can easily be captured by novices

who never have any experience with computers before. [23,32,61] have also been pointed out that the interaction techniques (such as icon, menu and dialogue box) used in model world interaction style, especially the direct manipulation, can reduce the cognitive efforts of users who are required to use computers to accomplish their tasks because users can manipulate the objects directly and they do not need as much energy as conventional world style UI requires them to interpret the computer output or translate their thoughts into commands that the computer can recognize. Consequently, model world style interaction increases the usability of computers for users.

Because of the advantages of model world style interaction to computer laymen, many companies foresee the potential marketplace of this style of interaction. Therefore, in the mid 80 and the beginning of 90, many commercial software products have been developed using this style of interaction, for example prototyping in Macintosh[79], Microsoft Windows and UI builder in NeXT computer[23]. It is believed that most modern UI designs are oriented in this interaction style.

2.3 Some Other Desirable Features and Problems of UI Development

Some desirable features and problems of UI development are identified in this section.

2.3.1 Separating UI from Application

2.3.1.1 Benefits of Separable UI and Application

Separation of UI and application is one of the key success of User Interface Management System (UIMS) [23,82]. There are several benefits:

1) Independent development.

As UI and application do not depend on each other, they can be developed separately without interfering with the other. We can develop and iteratively refine our UIs without considering the constraints from the applications. Therefore, we can accomplish the UI prototyping work more efficiently and shorten our development time.

In the other way round, we can also develop and iteratively refine our application core without considering its UI. By ignoring its UI, an application programmer can concentrate his/her effort on the logistics of the application. Hence, the capability and scope of the application can be enhanced.

2) UI can be personalized to the user.

For a particular application, users can choose their preferential UIs and "plug" them into the same application as shown in Figure 2.3

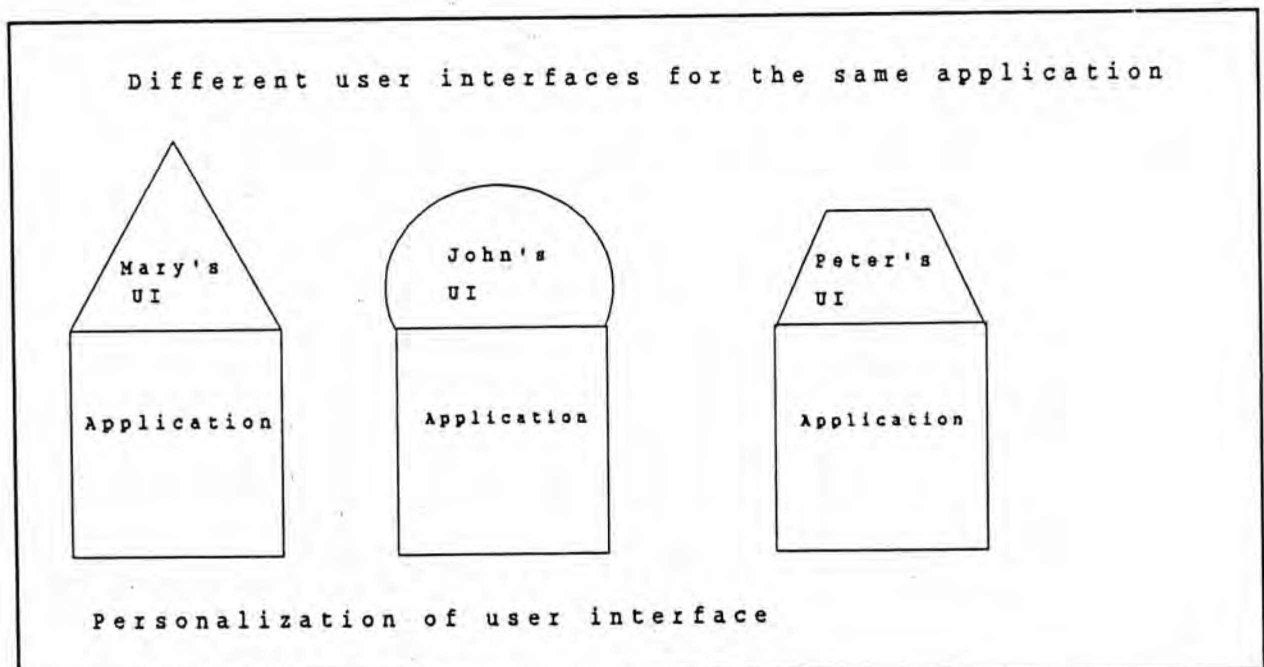


Figure 2.3 Personalization of user interface

3) Reusability

UIs can be reusable when they are "plugged" into compatible applications.

4) UI and application can be installed in different machines.

Advanced distributed computing technique, like the client-server model in many window systems, allow UI and application be installed and run on heterogenous machines as it is shown in Figure 2.4. By such a load balancing art, we can improve the efficiency and performance of the whole system. [51,88]

Although separation of UI and application can offer the above advantages, the following section shows that complete logical separation of UI and application is extremely difficult if not impossible.

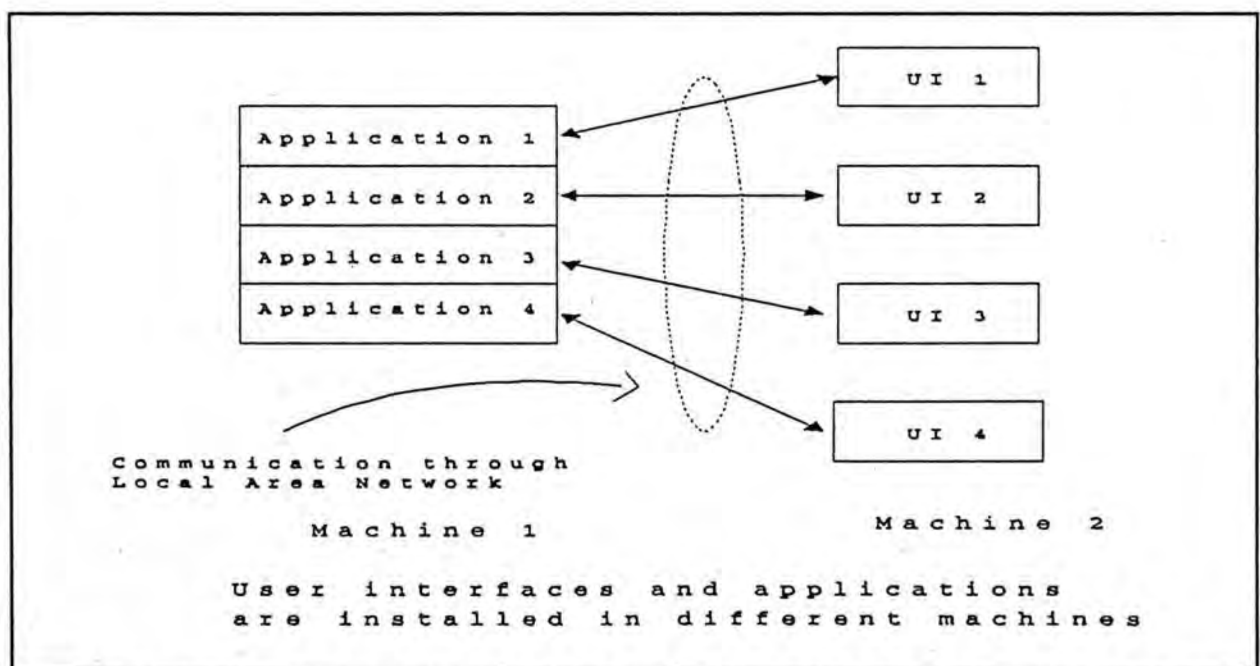


Figure 2.4 UIs and applications are installed in different machines

2.3.1.2 Requirements of Complete Separation

For a complete separation of user interface and application, we should have the following properties:

1) The behavior of each component does not depend on each other.
2) The status or configuration of each component should not be affected by each other.
3) Each component can stand alone without considering the existence of the other.

In order to achieve 1, neither of the user interface nor application can have control over the other. If one has control over the other, then the one being controlled will depend on the one who controls it. In this situation, we cannot develop UI and its application separately.

In order to achieve 2, neither the UI nor the application can access the data of the other (including referencing and changing the data of the other), that is no data dependencies of two components are allowed. If one can access the data of the other, status of one component can be changed by the other.

In order to achieve 3, UI and application should be mutually ignorant. It is the extent of point 2. Because of point 2, each of them even cannot access the other's status information. This point is crucial for independent development. Without knowing each other, neither of the development of the two components will be affected or constrained by the other.

The consequences of the requirements of complete separation is that the UI and application cannot communicate, reference, control or affect each other. Unfortunately, a system with such a complete separation of user interface and

application, as shown in Figure 2.5, cannot do anything for us as they cannot be integrated together to work for us.

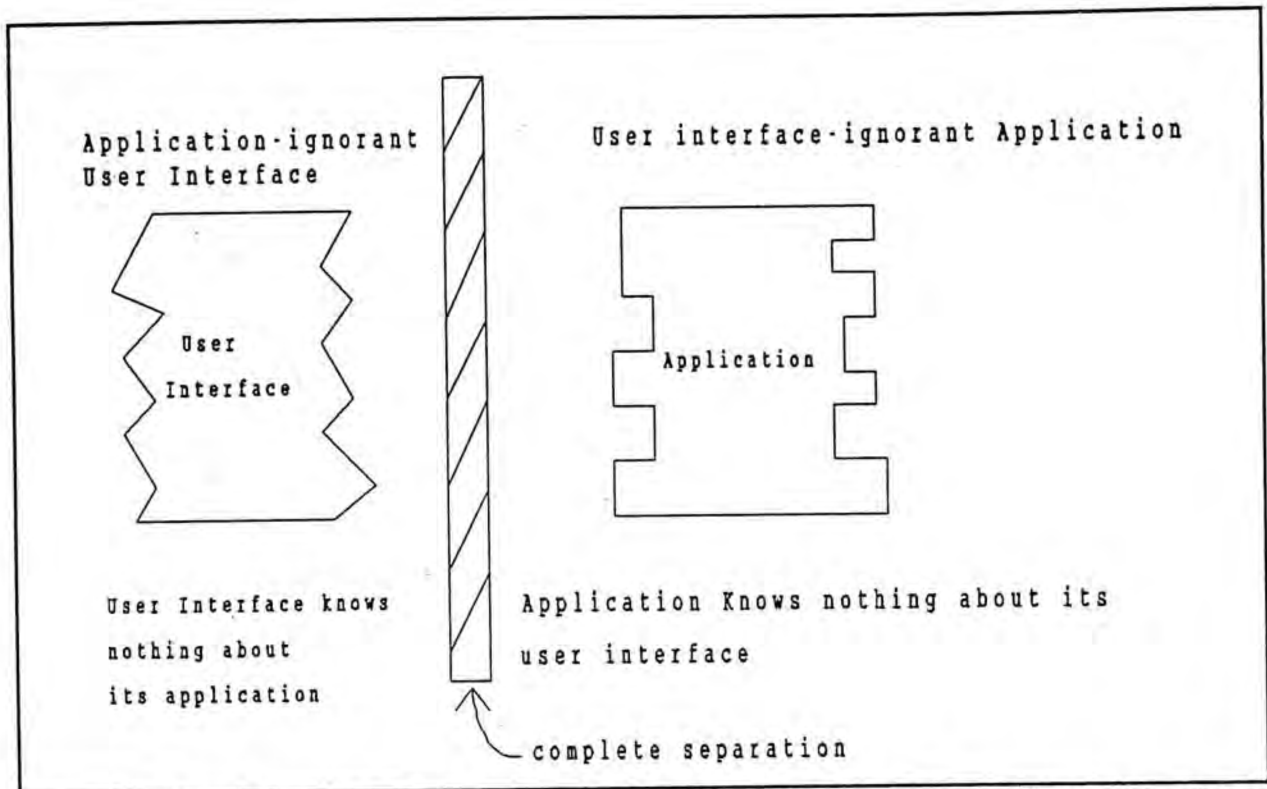


Figure 2.5 A system with complete separation of UI and application can do nothing for us.

Although a complete logical separation of UI and application is almost impossible, we can structure their designs at low level so that they can be implemented separately and independently. However, the dependence between UI and application at high level design specification is still inevitable.[57]

If we want to integrate the UI and application in an optimal way, we have to put more constraints in the design specification so that they can co-operate in the most efficient way. However, this will increase the dependence of user interface and application at the low level development and hence the flexibility for each component development will be decreased.

Therefore, there is always a trade off between

- the flexibility of component development,
- the independence of user interface and application

vs

- the efficiency of integrating the separable components
- co-operation among component, hence indirectly affect the whole system performance.

In short, we must maintain a balance between the system integration (at high level of abstraction view) and independence of the UI and application (at low level development view). In order to achieve this, we may need to find a methodology to formulate the design specifications so that we can enjoy the benefit of separable user interface and also the efficient co-operation between UI and application in the system.

2.3.2 Instant Continuous Feedback

As mentioned in section 2.1, instant continuous feedback is one of the features of model world style UIs. However this feature introduces new problems to world model type UI development.

2.3.2.1 Problems of Linguistic Model on World Model Type UIs

In a traditional linguistic model [23,32,40] for a conventional text-based interface, the user interface is viewed as a dialogue between a user and a computer. The model has three primary components at different levels as shown in Figure 2.6.

1. The component at lexical level consists of all input that will be recognized by a UI. It receives inputs from a user and checks if the input's tokens is valid such as correct identifier format or keywords.
2. The component at syntactic level consists of the syntax of the input command such as the number of arguments or the position of keywords in the command.

3. The <u>component at semantic level</u> consists of the knowledge about the meanings of the command. This component should be handled by the application and is outside the scope of the UI.

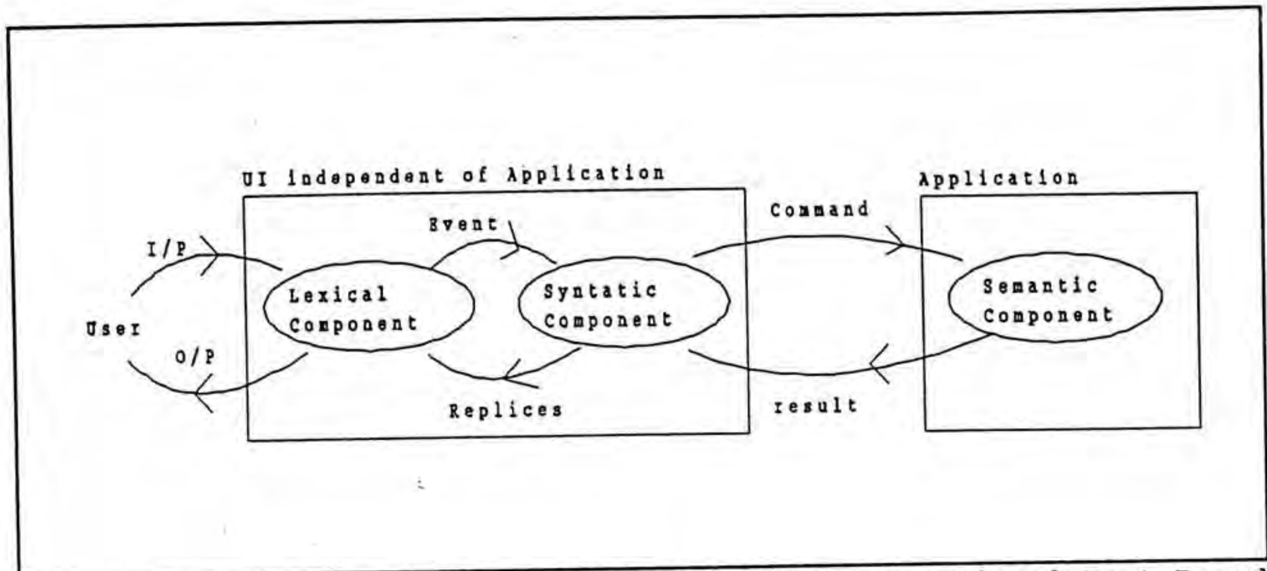


Figure 2.6 Traditional Linguistic Model for Conventional Text Based Interface.

These three components are strictly separated and should be independent of each other. The components at lexical and syntactic levels are embedded in UI while the component at semantic level is embedded in application. As these components are strictly separated, the feedback from these components are also separated and independent. Although instant lexical feedback can be supported by most operating systems such as echoing input characters, the syntactic and semantic components have to wait for complete input before they can give any feedback to the user.

Because of the above limitation, it has been pointed out that the linguistic model has problems on interactive graphics and direct manipulation interaction [51,57,82]. Usually, direct manipulation requires instant continuous feedback from all these three linguistic levels (lexical, syntactic and semantic). For example, in the Macintosh desktop environment, if we want to dispose a document, we need

to select an icon representing the document and drag it to a garbage can. This action requires lexical feedback by constantly showing the cursor location, syntactic feedback by changing the selected icon's position on screen, and semantic feedback by deleting the document in the file cabinet and showing the increased size of the garbage can. From the above example, we can see that in order to provide direct manipulation interaction, the three linguistic components should no longer be separated. They collapse into one single entity as each of them requires help from the other in order to give immediate feedback to end users. Continuous feedbacks from all linguistic components will be given to users even though the users may not have finished their commands.

However, without the linguistic model, we may run out of an effective way for describing the essential dialogue control and events sequence of UI. Hence, in order to model the model world style UI, a new model is required such that it can describe the instant continuous feedback mechanism from the three levels and at the same time can capture the dialogue control and events sequence of the UI as well.

Instant semantic feedback also further complicates the problem of separating UI from application mentioned in section 2.3.1. If we consider application as a kind of semantic server as most UIMS models do [18,40], in order to have efficient semantic feedback, we may need to build more semantic knowledge in a UI. However, this will increase the dependence between UI and application at low level design. On the other hand, if we provide the semantic feedback by establishing closer communication between UI and application, it will increase the dependence between UI and application at high level design and also

the loading of communication between them. Therefore, there is also a trade off between separable UI and efficient semantic feedback.

2.3.3 Undo and Recovery

No one can guarantee that one will never make any mistakes when one is using a computer. Therefore, it is better for users to undo their previous actions and cancel the effects that they have just made. With this feature, UIs become more friendly and forgiving to users. Users can also feel easier to operate their systems as they can return to the original status in case they take a wrong action.

Sometimes users may also want to backtrack several steps in order to try different paths to accomplish their tasks. Hence, the undo feature also gives users more power to solve their problems by taking different alternatives.

Nevertheless, unlike conversational style UIs, the undo unit in Model world style UIs is not as clear as conversational style UI. In conversational style UI, a single character can be considered as an undo unit as it can be "back up" to the previous state by using a backspace key. However, in model world UI, undo units are not well defined. A single operation in model world style UI may involve several actions. How many steps should be backed up in order to undo an operation? In model world style UI, feedback is continually given to a user even when they have not finished their input. When we undo an operation, all the effects from the feedback which associate with the operation should also be canceled. The situation is further complicated if the operation involves nested closure which may cause difficulties when an undo action is required in the most inward nested closure. Therefore, the overhead of a undo operation in Model

World style UI is higher than a undo process in conversational style UI.

2.3.4 Iterative Design through Rapid Prototyping

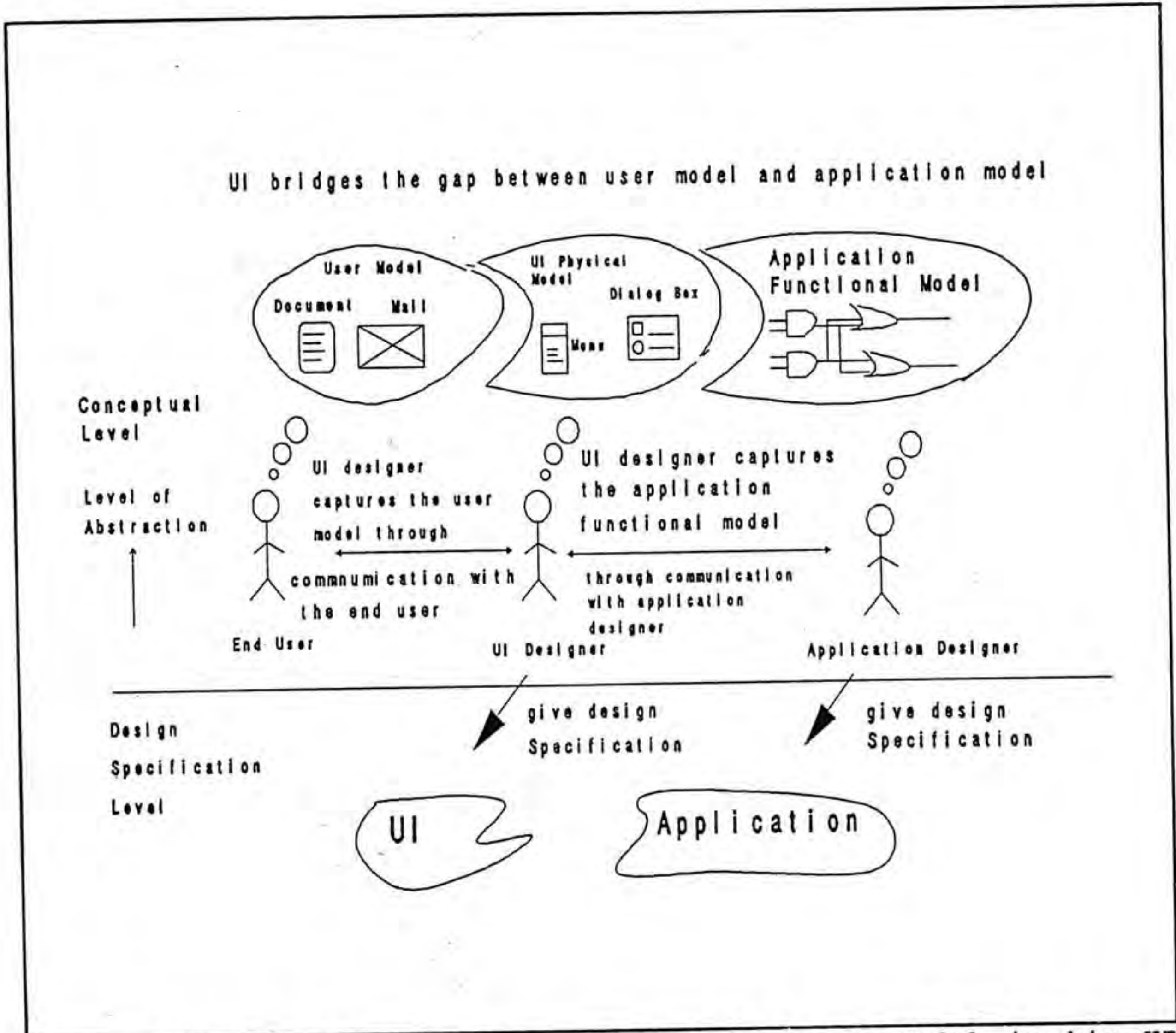


Figure 2.7 UI designer has to capture the user model in his UI design.

As a UI is considered as a physical bridge between users' world and their computer system world (as shown in Figure 2.7), in order to design satisfactory UIs for users, UI designers have to understand their client user models, so that they can develop the right products that really fit their client needs.

The problem of capturing user models and mapping the models to computer system world may involve several different disciplines such as cognitive science, ergonomic science, human behavior and human-computer interaction. At

the very beginning, users may express their original intentions and requirements by drawing, typically on paper, scenarios of how the user interaction will look and act. Then, by studying the acceptance test results of the UI prototypes and feedback from users as shown in Figure 2.8, UI designers will be able to understand their clients' desires better. Multiple iterations of design and refinements of UI are, therefore, necessary before a satisfactory UI is built.

Bedsides helping UI designers to discover early design errors, prototyping also help end users to discover their unknown needs. At the beginning, they may not clearly know their actual needs of their system, they just have a coarse idea of their needs. However, as they cope with the UI prototype, they can really visualize their needs and make their requirement become more concrete. Hence, prototyping also assists the requirement analysis in the UI development life cycle

In order to achieve rapid prototyping, a new software development approach should be proposed for UI development so that the modification of UI software becomes easy and efficient.

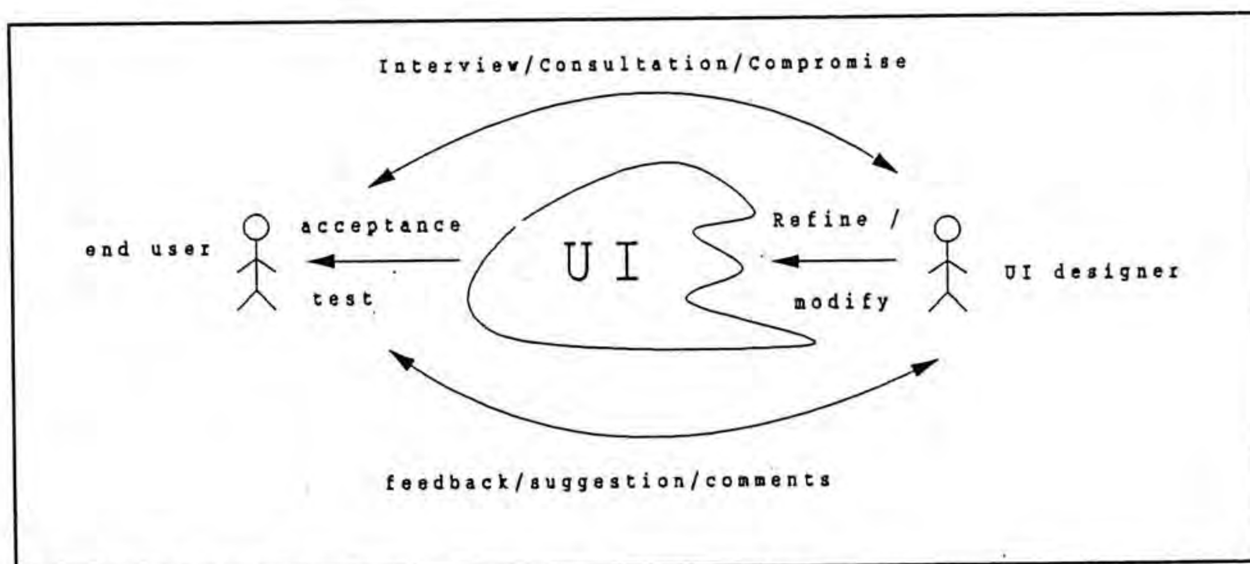


Figure 2.8 Multiple iterative design may be necessary through the user feedback loop

Chapter 3

An Object-Oriented Model for User Interfaces Development

In this chapter, an Object-Oriented Model for UIs development is proposed. This model tries to provide easy mechanisms for supporting separable UI, multiple feedbacks, undo functions and multi-thread dialogue features in modern UIs. A new notation for UI specification is also proposed in this chapter.

3.1 Features of UIs to be supported by the Model

In section 2.3, we have presented the desirable features of UIs and some problems of their development. Based on these features and problems, the proposed model should meet the following criteria.

1) Dialogue independence.

Dialogue independence is a UI design approach in which design decisions affecting only the human-computer dialogue are isolated from those affecting only application system structure [23,32,33]. That is an application should know nothing about interaction styles (e.g. using menus, buttons via mouse or command languages via keyboard) and appearances (e.g. the presentation of data to users such as table, graph or chart) of a UI. Conversely, the UI knows nothing about how the application processes its requests. Dialogue independence is a basic foundation for separating UI from application. Without this feature, the computation functions in application will merge into UI and makes the separation of UI and application become difficult.

2) Multiple Continuous feedbacks from three linguistic levels (Lexical, Syntactic and Semantic).

The proposed model has to cope with the problem of merging the three linguistic levels described in section 2.3.2.1. The model should provide mechanisms to give feedbacks to a user but at the same time provides straightforward syntactic mechanisms for describing events sequence of UIs such as those provided by most linguistic models.

The model should also deal with the balance between UI separation and communication overhead between UI and application due to semantic feedback as described in section 2.3.2.1.

3) Multi-thread dialogue.

The model should provide a new notation to describe multi-thread dialogue control in model world style UIs. Besides, describing the events sequence in human-computer interaction, this new notation should also capture the mechanisms for communication and consistency among different independent dialogues.

4) Undo function.

The model should provide a simple and efficient mechanism to handle undo functions in UIs regardless of the abstract levels or the complexity of the undo functions.

3.2 A Linkage Model for Separating UI from Application

It has been shown that in section 2.3.1.2, a complete logical separation of UI and application is impossible. However, we need to integrate them into a

香港中文大學圖書館藏書

system so that it can work for us and on the other hand we want to preserve the independence of UI and application as much as possible. At least, the dialogue independence can be achieved. Consequently, in order to solve the above conflicts, a module called Linkage, which is shown in Figure 3.1, is introduced. Linkage is used as a mediator to link up a UI and its application. As linkage has knowledge about both the UI and the application, it can translate messages for them and hence lets them communicate with each other indirectly through itself.

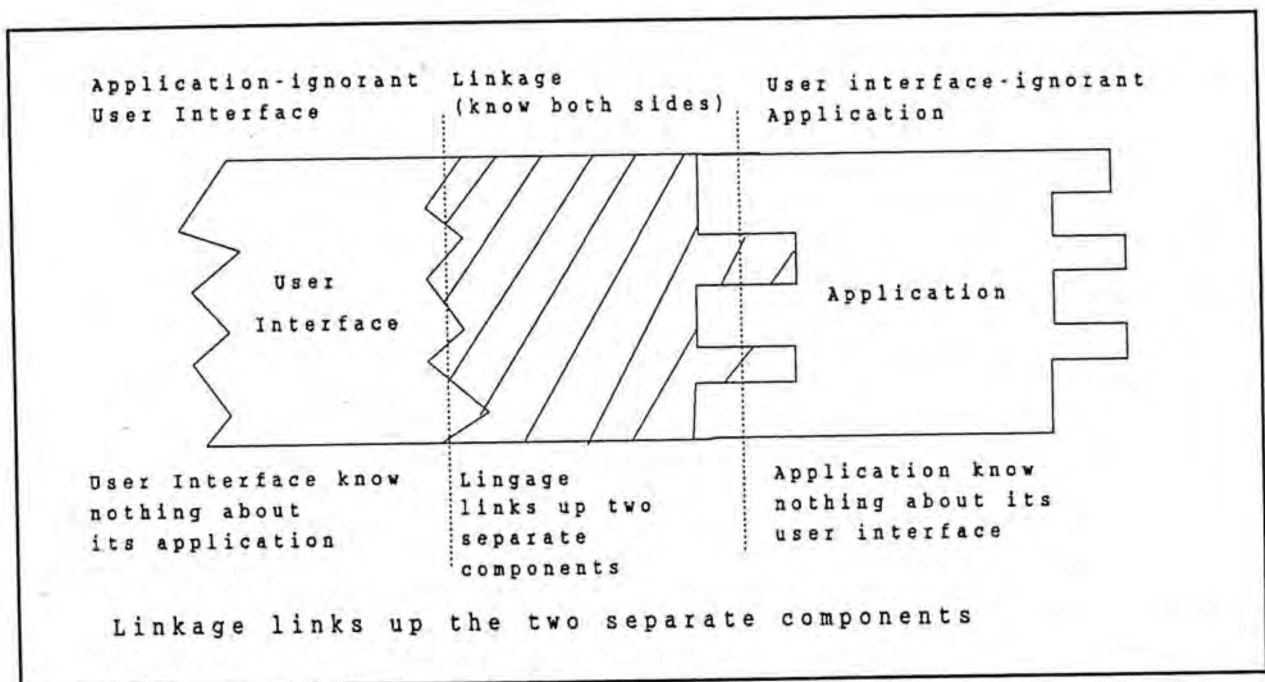
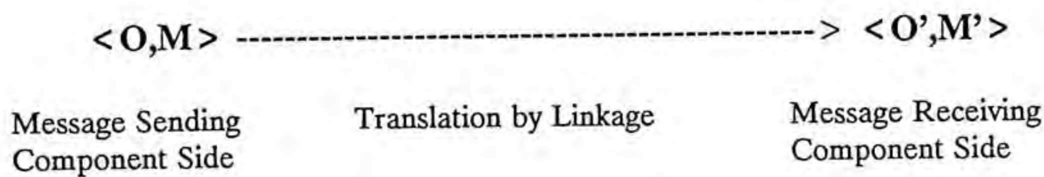


Figure 3.1 A Linkage component

3.2.1 Communication Messages Modeled using Object Oriented Approach

As Linkage is designed to preserve the mutual ignorant properties of UI and application to a maximal degree, indirect communication between UI and application is defined in a highly abstract fashion as abstraction helps the developers of both UI and application to ignore the implementation details and constraints due to their own interactions. Abstraction, of course, allows the semantics of system objects to be embedded in messages too.

In [24,58,62], it has been pointed out that the mechanisms of generalization, specification, inheritance, classification, aggregation, and encapsulation of method and data in object oriented paradigm can help us to capture the semantics of objects in a system. Therefore, in our Linkage model, we define each message for communicating between the UI and the application to be an order pair **<O,M>**, where **O** is an object to be manipulated and **M** is the method that applies to **O**. The translation of a message is actually a mapping from **O** and **M** to **O'** and **M'** according to the interface specifications of the UI and the application.



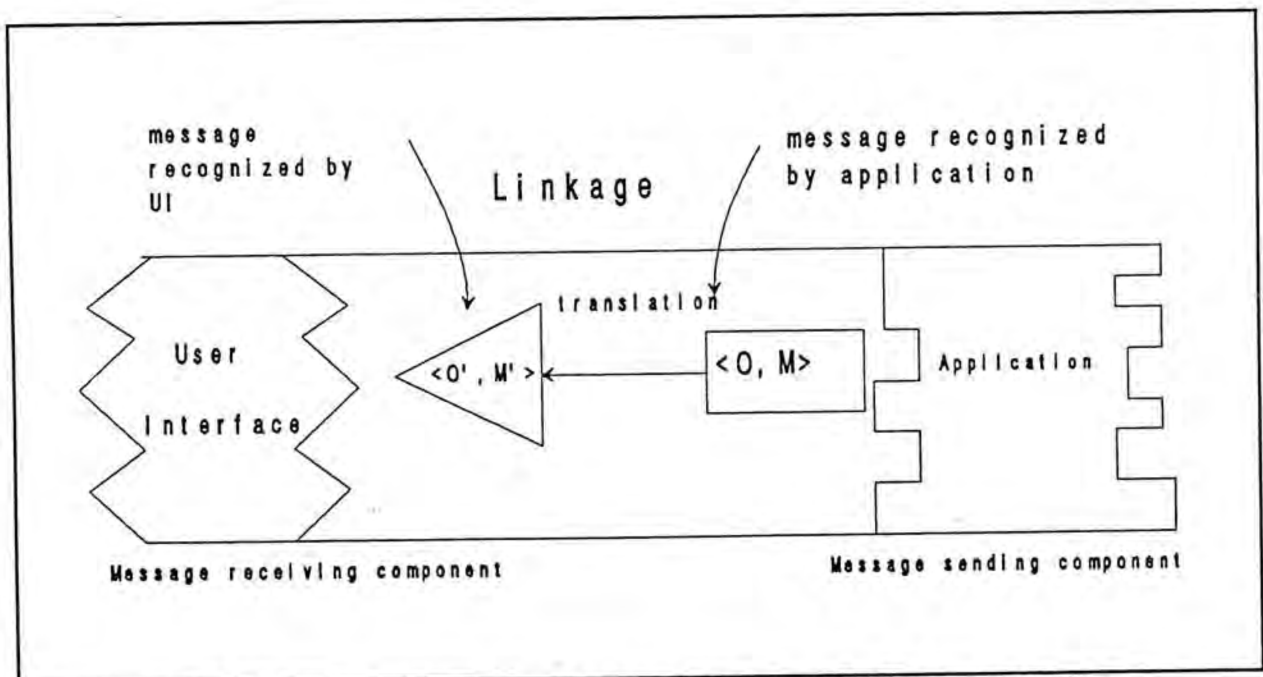


Figure 3.2 Linkage translates message from <O,M> to <O',M'>

O is the object requested by a Message Sending Component (MSC) for manipulation by a Message Receiving Component (MRC). UI or application can either act as MRC or MSC exclusively. **O** should be organized and composed by

the MSC in such a "favorable" way for easy or convenient manipulation by the MSC. **M** is the method requested by the MSC for application to the object **O** by the MRC.

As the MSC knows nothing about the MRC, message **<O,M>** can be designed in a highly abstract manner. Actually, the order pair **<O,M>** can be viewed as an "intention" (**What** to do), something that we want to do but not "**HOW** to do", of the MSC. By conceptualizing a sending message as an intention, the developer of the MSC can ignore the actual detail processes and implementation in the MRC.

O' is the object translated from **O** by the Linkage. It can be recognized by the MRC. Similarly, **O'** is in a form such that it is "favorable" for manipulation by the MRC. **M'** is the method applying to **O'**. It is translated from **M** by the Linkage for recognition by the MRC.

Obviously, the mapping of **<O,M>** to **<O',M'>** may not be one-to-one. It may be one-to-many or many-to-one. If the mapping is one-to-many or many-to one, Linkage has to take actions to decompose or group message(s). These actions may include filtering unnecessary information and collecting necessary information for both sides before the Linkage sends the translated messages to MRC.

3.2.2 <u>A Sample Message</u>:

MSC	: Application
MRC	: User interface
Intention of the MSC	: "Display a chart to user"

<O,M> : **<Table,Display>**

<O',M'> : **<Table',Display'>**

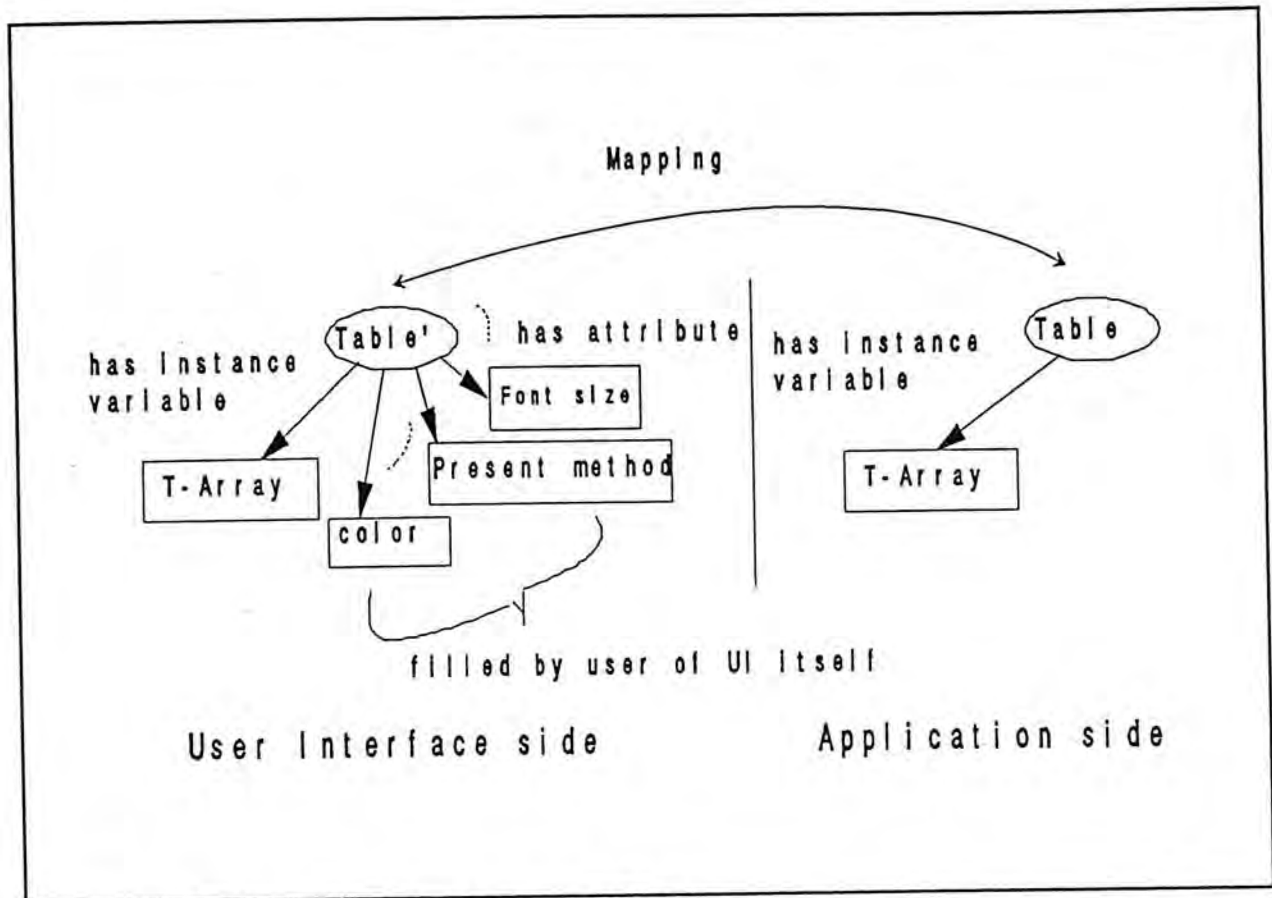


Figure 3.3 Mapping of Table to Table'

Table is an object containing an array variable, **T_Array**, which stores a chart's data. **Display** brings out the intention of the MSC but the way to present the chart is left to the UI. This is the concept of dialogue independence. In addition to **T_Array, Table'** object also contains some attributes for presenting the chart as shown in Figure 3.3. **Display'** is the method applicable to the object **Table'**. Its task is to handle the presentation of **Table'**. As the application has not specified the way for presenting the chart, the UI can let users choose their own presentations of **Table'**. According to the choice of user, **Table'** may be one of the following objects:

Poly_Table' : present Data in polygon chart when receives Display' message

Hist_Table' : present Data in histogram chart

Bar_Table' : present Data in bar chart

Graph_Table' : present Data in graph

spd_sh_Table' : present Data in spread sheet

By making use of polymorphism in object oriented paradigm, different objects present the chart differently with the same **Display'** message as shown in Figure 3.4.

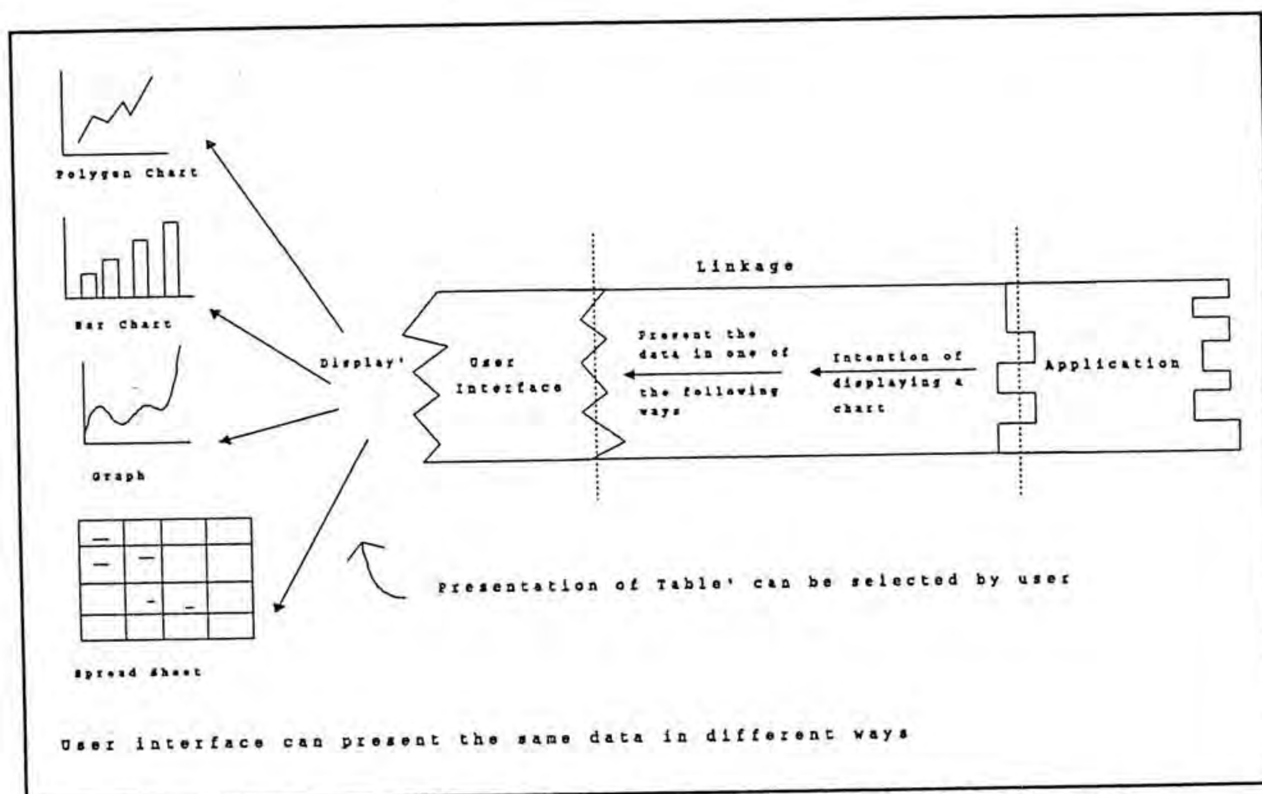


Figure 3.4 User interface can present data in different way

3.2.3 Linkage in A Distributed Heterogenous Environment

The Linkage model can be extended to a distributed heterogenous environment in which each communicating component (UI or application) possesses its own Linkage element, as shown in Figure 3.5. Message communication between the two machines can be implemented by Remote

Procedure Call (RPC) [5]. Moreover, the concept of stub [3,25] can also be embedded in the Linkages of both sides.

Since the architectures and data representations of machines at UI side and application side are different in a heterogenous environment, the mapping of messages must also include data format conversion capability. Therefore, the Linkages of both sides also take care of packing and unpacking of outgoing and incoming data respectively.

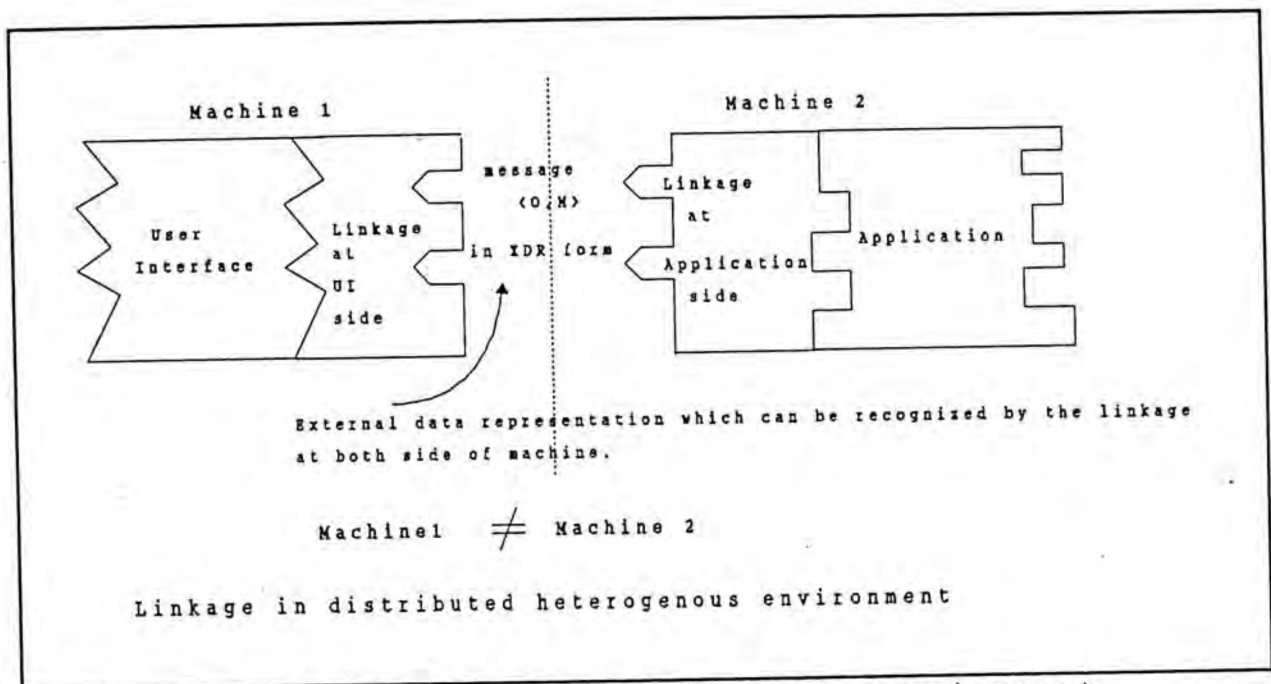


Figure 3.5 Linkage in distributed heterogenous environment

3.2.4 Comparing the Linkage Model with the Application Interface Model in Seeheim's UI Model

The functions of Linkage is quite similar to the application interface model in the Seeheim UI model [8,29,40]. Both of them are considered as mediators between UI and application and also support dialogue independence. However, the Linkage model views the UI and application in term of objects rather than application routines and data structures. The interaction style of UI and

application in Linkage model is based on their message communication mechanism rather than routine calls from each separate component. This interaction style encourages mixed control structure in dialogue control in which neither UI nor application has control over the other. Although the two models view and interact with UI and application differently, application interface model in Seeheim model actually provides a good model foundation for Linkage model.

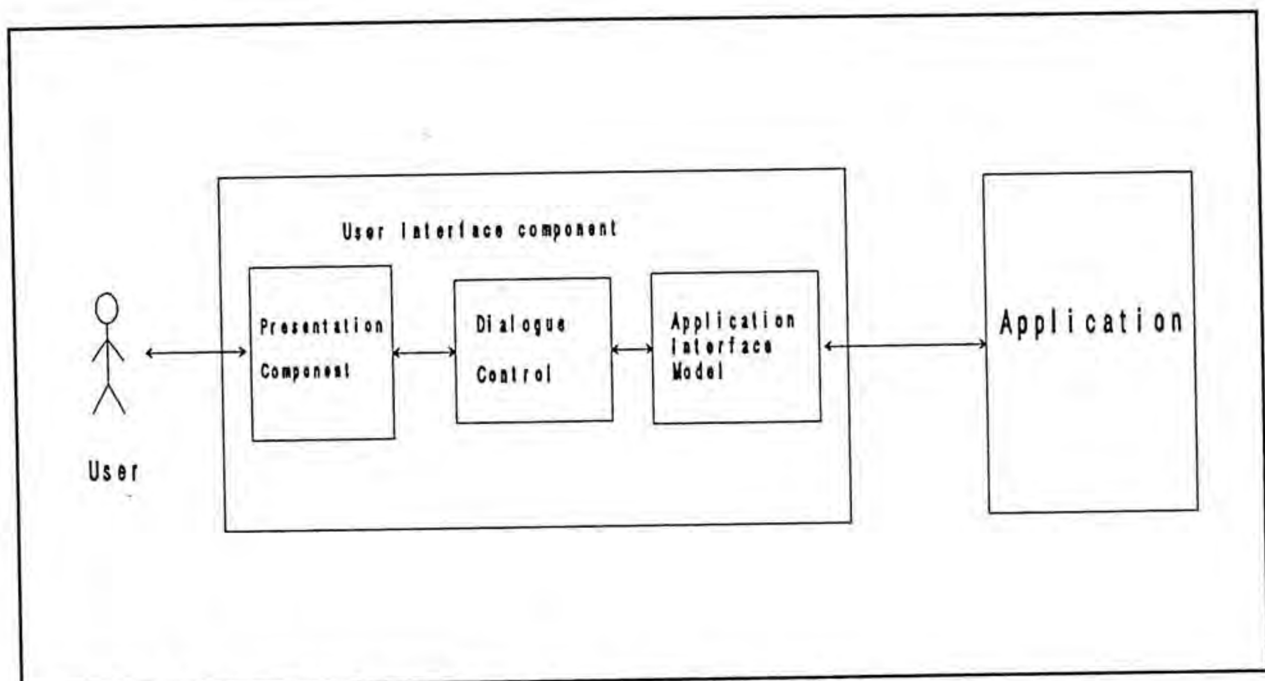


Figure 3.6 The Seeheim model of user interface

3.3 An Object-Oriented Model for Supporting Multiple Feedback and Multi-thread dialogue

In order to satisfy the criteria of 2 and 3 mentioned in section 3.1, an Object-Oriented model is proposed. This model is modified from the linguistic model shown in Figure 2.6. However, unlike traditional linguistic models which have only one set of lexical component, syntactic component and semantic component, the proposed model allows several components to be located on a linguistic level. Each component in the model corresponds to an object in Object-

Oriented paradigm. Objects in the proposed model are no longer strictly separated and independent of each other as traditional linguistic models do. In addition, some objects are located on an overlapped regions of two linguistic levels (semantic, syntactic or lexical) so as to support multiple continuous feedback and errors checking for each level.

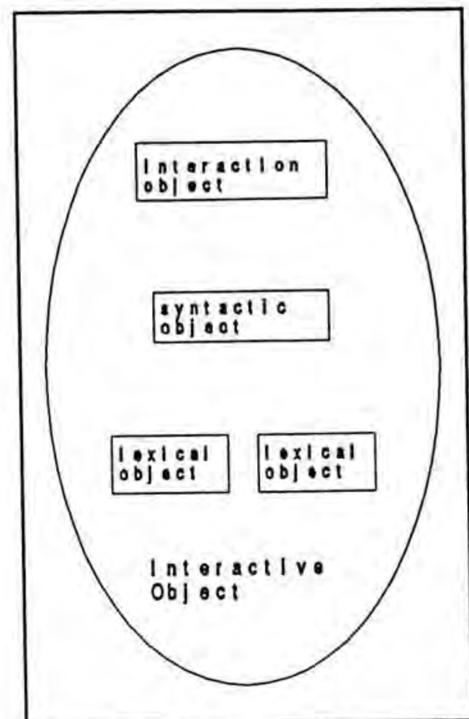


Figure 3.7 An interactive object can consist of a set of interaction, syntactic and lexical objects

A set of objects which may be located at any one of different linguistic levels forms an interactive object (shown in Figure 3.7) with which a user can conduct a dialogue and apply direct manipulation. For example, memos or invoices can be interactive objects as a user can manipulate them directly and can conduct meaningful dialogues with them. Actually, in the multi-thread dialogue paradigm, dialogue switching is performed among interactive objects. The sections 3.3.1 to 3.3.5 explain the possible inner objects in an interactive object. The schematic representation of the relations among these inner objects at each linguistic level in an interactive object is shown in Figure 3.8.

3.3.1 An Overview of the Model

A UI can contain several interactive objects and each interactive object in turn can contain several other inner objects such as lexical objects, display objects, presentation objects, syntactic objects and interaction objects.

- Lexical objects are responsible for lexical parsing of user's inputs. All the user's inputs are supposed to be routed to these lexical objects first. They can also give instant lexical feedback to the user through display objects.

- Display objects display information according to the messages from lexical objects or presentation objects . They usually contain some standard graphic library functions.

- Presentation objects determine how information in an interactive objects is to be presented to a user. For example, a presentation object determine the display format of the information.

- Syntactic objects check the "grammar" of the user's input and give appropriate feedbacks if necessary.

- Interaction objects determine how to interact with the user. The main human-computer interaction dialogue is embedded in these objects. These objects are also responsible for communicating application through the Linkage.

Besides the above objects, a UI also contains an Interaction Knowledge Base. This Interaction Knowledge Base contains all the current global states of the UI such as which interactive object is activated or deactivated. Communication between interactive objects can be performed through this Interaction Knowledge Base.

3.3.2 Objects on the Lexical Layer

A Lexical Object at the lexical level is responsible for instant lexical feedback and lexical parsing. The listener shown in Figure 3.8 receives all input

messages from input devices. It then passes them on to the appropriate Lexical Objects according to the current state of the system and the type of the message. If the input message only requires a simple lexical feedback such as changing a cursor shape, the listener will forward a message to a Lexical Object. The Lexical Object will respond the received message by sending another message to a Display Objects which will give appropriate feedback to a user through a set of output drivers. Lexical Objects can only handle the message at lexical level. Other message rather than at this level, such as syntactic checking, will then be passed to Presentation Objects.

3.3.3 Roles of Presentation Objects and Display Objects

The responsibility of the Presentation Objects is to determine how information in an interactive object is to be displayed to a user. As it is located in both the lexical and syntactical regions, it can perform partial syntactic checking before it forwards an input message to Syntactic Objects or Interaction Objects. A combination of lexical and syntactic feedback can be preformed through the Presentation Object. For example, using a mouse to select an icon object and dragging that object to a certain valid position requires continuous feedback from both lexical and syntactic level. Selecting an object may require a lexical feedback to change the shape of the selected object so as to indicate that the object is successfully selected. Dragging the object through a window area may require a syntactic feedback to show the current position of the selected object.

Using the above example, when a user presses a mouse button to select an object, the "button pressed" message will be sent to a Lexical Object through the

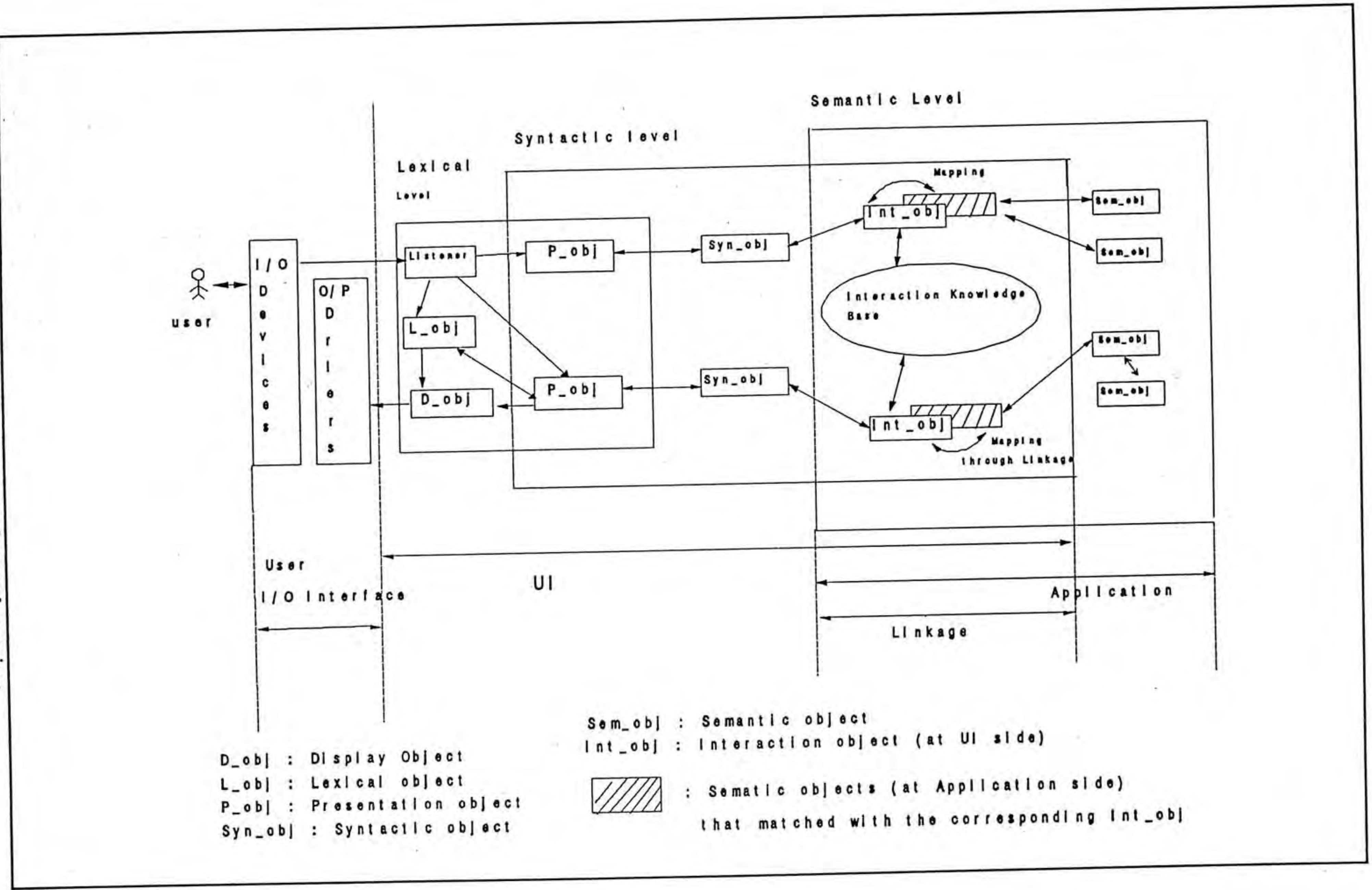


Figure 3.8 Some objects are in the overlapped region

listener. Then the Lexical Object changes the shape of the selected object through a Display Object. When a user moves a mouse to drag the selected object around, the "mouse moved" message once again are forwarded to the Lexical Object but this time the Lexical Object cannot handle this message alone as it requires syntactic checking. Therefore, in addition to giving an immediate lexical feedback, it also passes this message to a Presentation Object. The Presentation Object checks the valid mouse location, such as relative position to other object, and shows the current object position through the Display Object. The above feedback is given to the user continuously until the user releases the mouse button.

The task to present information is the job of Presentation Objects. The duty of the Display Object is to display the information according to the message from Presentation Objects or Lexical Objects. A UI can have more than one Display Object. Display Objects can pass messages with each others and can form an object hierarchy. Display Objects at the lowest level contain a collection of some primitive functions such as erasing a screen, drawing a line, a circle etc. Through these functions, Display Objects can utilize some standard graphic library functions.

3.3.4 Syntactic Objects

At syntactic level, Syntactic Objects receive messages from Presentation Objects for syntactic checking and feedback. If the "grammar" of the receiving command is correct, such as selecting the right objects, moving them to valid positions and making right connections with other objects, a Syntactic Object will

give immediate positive feedback, if any, and forward the command to an Interaction Object; otherwise, it gives negative feedback to user through the Presentation Objects.

3.3.5 Interaction Objects

Interaction Objects receive messages from Syntactic Objects or directly from Presentation Objects and then perform semantic checking and give semantic feedback if necessary. If Interaction Objects can handle the command without the help of Semantic Objects in application, they will give feedback to a user through the Presentation Objects; otherwise they will forward the command to the Semantic Objects through the linkage component. Therefore, Interaction Objects determine whether a UI needs to communicate with its application or not. Actually, Interaction Objects can be considered as images of Semantic Objects on the application side. They have some general knowledge of the application semantics, but the implementation of these semantics rests wholly on the application side and is hidden from the Interaction Objects.

Occasionally, Interaction Object can give semantic feedback to a user through Presentation Objects without going through the Semantic Object at the application side. For example, an Interaction Object may send message to a Presentation Object to give warning message if some input data values are out of range or too low. Of course this semantic information has to be embed in the Interaction Object first.

3.3.6 Interaction between Objects and Linkage Component

When an Interaction Object on the UI side want to pass a message to a Semantic Object on the application side for services, the message will be passed through a Linkage Object which translates the message to a form recognizable by the Semantic Objects on the application side. The Semantic Objects which receive the message may continue to forward the message to other appropriate Semantic Objects if necessary.

The Linkage object performs a mapping of Interaction Objects to Semantic Objects and vice versa, so that the Linkage object can figure out which objects on the receiving side should receive the message. The mapping may be one-to-many or many-to-one. It is determined by the difference between the data structures and object hierarchies on both sides.

In order to support dialogue independence, a message sent by Semantic Objects to Interaction Objects should not contain any human-computer interaction dialogue control information because Semantic Objects are supposed not to take care of any interaction with a user. In the view point of an application designer, the Semantic Objects only receive error free input data, work out correct results and then send the results back to UI. How a user interacts with interactive objects is the job of UI. After the mapping by the Linkage component, it is the responsibility of UI to take care of the dialogue with the user.

3.3.7 Multiple U-tubes Ladder for Supporting Multiple Feedback

A whole interactive object can be considered as a multiple U-tubes ladder as shown in Figure 3.9. The inner objects are distributed in different U-tubes. The

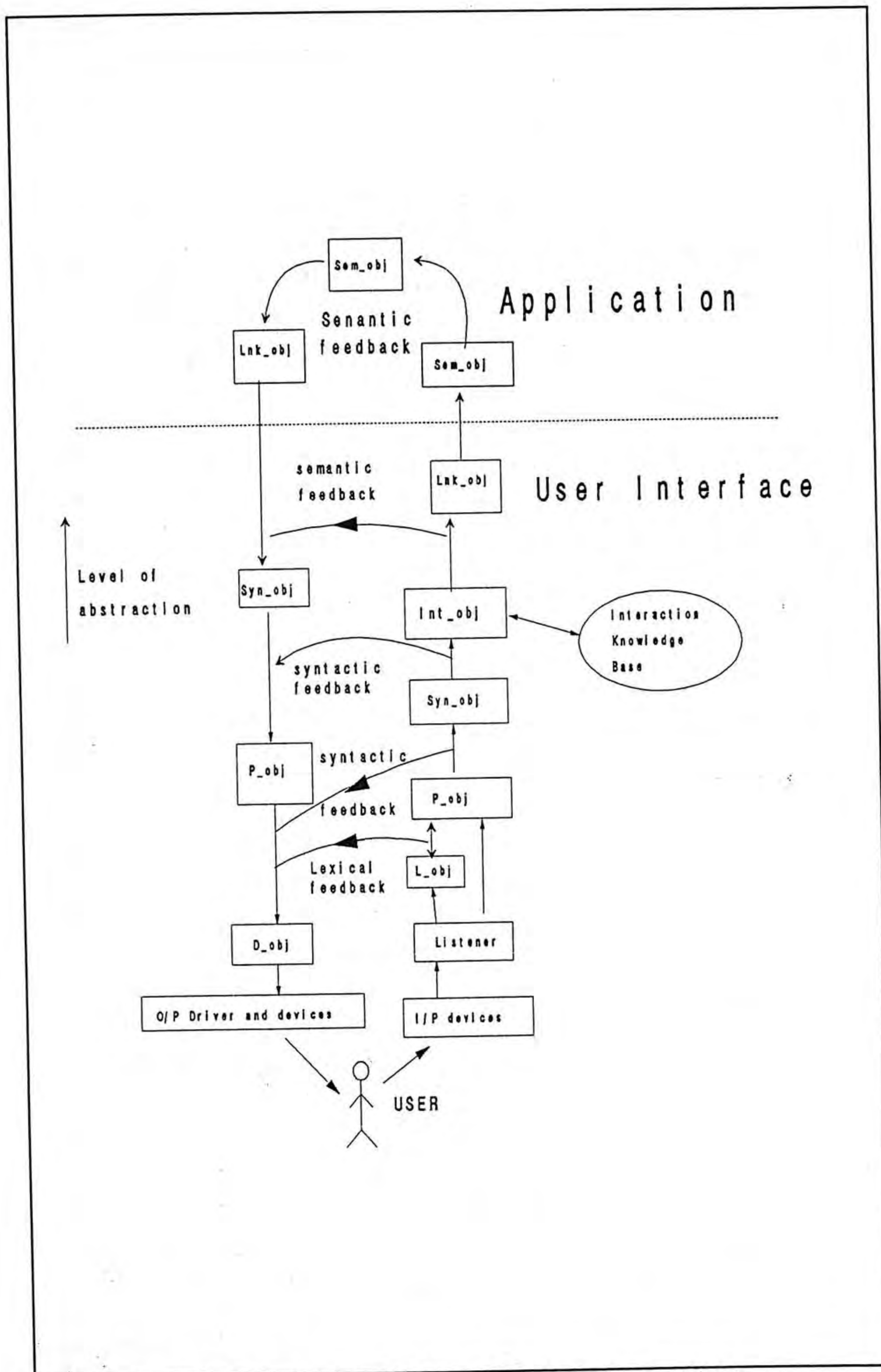


Figure 3.9 Multiple U-tubes ladder in interactive object

level of abstraction decreases from higher to lower portion along the ladder. The upper portion of the multiple U-tubes ladder corresponds to Semantic Objects in application while the lower portion corresponds to the Lexical Objects in UI. A feedback can be given through one of this U-tubes. However, multiple continuous feedbacks can also be given from objects located at different abstraction levels simultaneously. In fact, feedbacks from different U-tubes can be given to a user along the multiple U-tubes ladder simultaneously. If an object in the U-tube cannot handle the input command, it will pass the command to its upper U-tube objects and let them handle it. They may give immediate feedbacks or/and pass them to their upper U-tube objects.

3.3.8 Recovery through a Generic UNDO Stack

An UNDO operation in a modern UI is more expensive than conversational world style UI because besides requiring the previous status of the operation to be restored, it also requires the effects from the feedback which associate with the operation to be canceled. Interactive objects can restore their previous status by retrieving the past history stored in a undo stacks which is updated constantly by

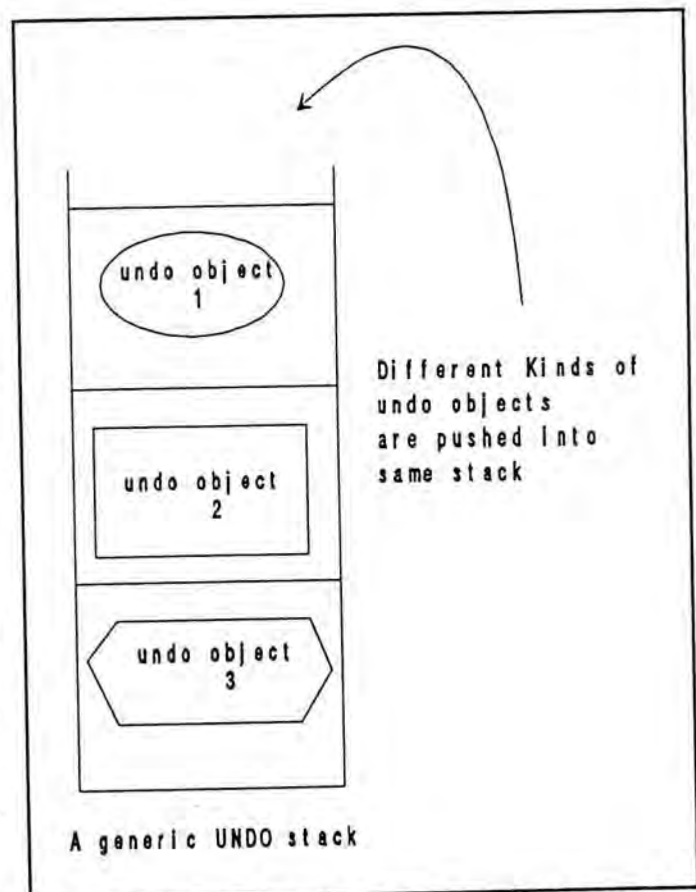


Figure 3.10 A Generic UNDO Stack for Recovery

their inner objects whenever they perform any reversible operations. However, sometimes it may become complicated as a UI may also need to determine which part of the previous status should be restored and which part should be kept unchanged. In addition, restoring only the previous status may not be good enough, as the effects caused by the previous operation have to be canceled out too. For example, if an operation cause the side effect of displaying some warning messages, these warning messages have to be erased as we undo the operation. In these cases, further actions have to be taken to cancel out these effects. Instead of pushing the previous state into the undo stack, an undo object is pushed into the undo stack. Besides the previous state, an undo object also contains a series of reverse functions necessary to cancel out the effects caused by the previous actions. The undo object also determines which state should be restored and which state should be kept unchanged according to the nature of the undo operation. The content of an undo object is transparent to an interactive object. Whenever a undo operation is required, the interactive object pops a undo object from the undo stack no matter what type of undo object it is and then let the undo object take care the rest of the job.

The undo stack should be shared by different types of undo objects regardless the complexity or the abstract level of undo operations. Hence a single undo stack is enough for different types of undo operations in an interactive object. The polymorphism and dynamic type mechanism in object oriented paradigm are used to construct this generic undo stack. The implementation of this generic undo stack is further described in section 5.1.3.

Any UNDO operation that requires semantic knowledge in application will

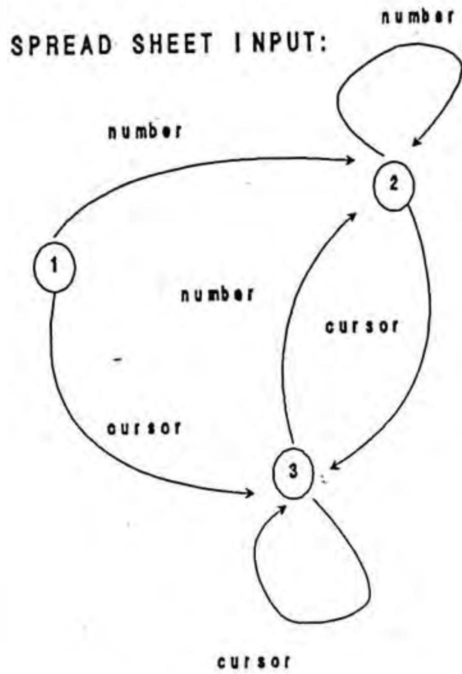
be passed to the Semantic Object at the application side through a Linkage object.

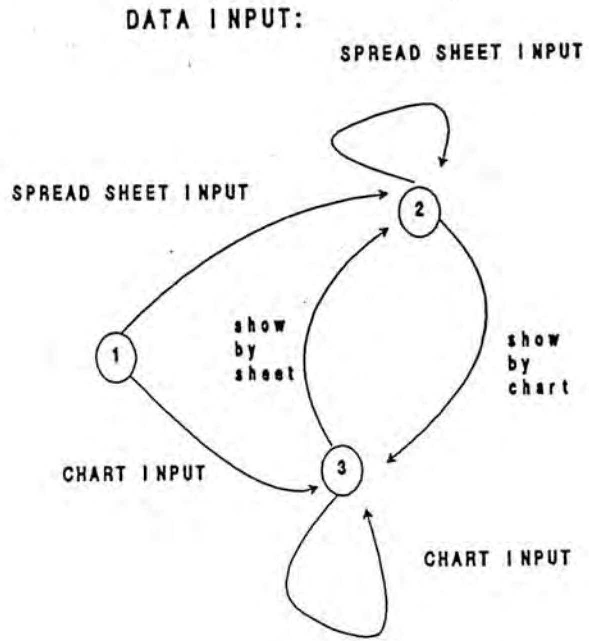
3.3.9 Dialogue Control within Each Object

Unlike a traditional linguistic model which has only one large dialogue control between a user and a computer system, the dialogue controls in this model are distributed among objects on each level. As it has mentioned in 3.3.6, in order to have multiple feedbacks, objects at different abstraction levels, have responsibility to give feedbacks to a user. Therefore, each object should have its own dialogue so as to respond to user inputs and give appropriate feedback to users. For example, a lexical feedback for a toggle button may need a Lexical Object to remember the previous state of the button. The dialogue control in each object can be modelled by a finite state automata which can memorize the user input sequence and is well described by a transition network.

Although the dialogue controls are distributed among objects, it does not mean that they are independent of each other. The top abstract level dialogue control is in an Interaction Object. This is the main dialogue control of an interactive object is in the Interaction Object. If a transition network is used to describe this level dialogue, then an arc label of the network may correspond to a lower abstract level dialogue control which may be embedded in Presentation Object or Lexical Object. That is an arc label in a higher abstract level dialogue can be considered as pointer to another dialogue in a lower abstract level or vice verse.

The dialogue controls shown in Figure 3.11 are described by transition





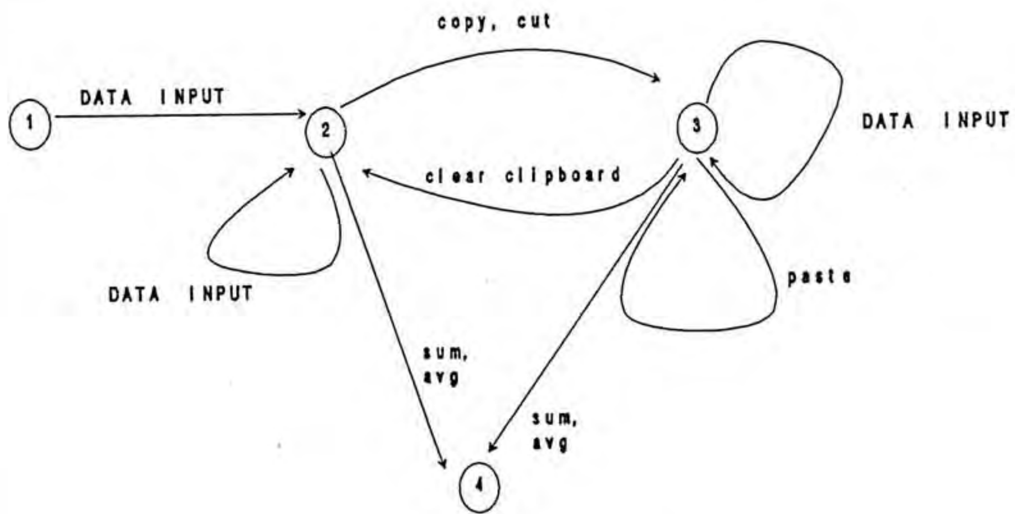


Figure 3.11 Dialogue control in an object

networks. Each arc label in the transition network corresponds to a token in the dialogue control. A token is the smallest meaningful unit in the dialogue control. It can correspond to a single incoming message (indicated by lower case letter) of an object or other level dialogue control (indicated by upper case letter) in other object.

From page 47, an application is designed to calculate the sum of sales in a week and the average of sales each day. Its UI can present the sale data either by chart or by spread sheet. A user can input the sale data either through keyboard if the sale data are presented in spread sheet form or through mouse if the sale data are presented in chart form. This UI also allows a user to copy sale data into a clipboard and to paste them to somewhere else later when necessary. Figure 3.11 illustrates some examples of dialogue control in Lexical, Presentation and Interaction Objects.

3.3.10 Interactive Object

As mentioned in the beginning of section 3.3, an Interactive Object may consist of clusters of inner objects including Interaction Objects, Presentation Objects, Lexical Objects and Display Objects. Moveover, an interactive object must have at least one or more Interaction Object. Besides the above inner objects, an interactive object also contains a cluster controller to connect an active message path among its inner objects according to the current states of the interactive object. All incoming messages from a user to this interactive object will be routed to the inner objects through this active message path as shown in Figure 3.12. In addition, an interactive object can also have its own undo stack to

perform an undo action. Within a cluster of inner objects, there is only one Interaction Object but the cluster can have several Presentation Objects or Display Objects.

As an Interactive Object can have more than one cluster of inner objects and each cluster of inner objects have their own dialogue controls , by connecting different message path among the inner objects, it can support different interaction styles.

3.3.11 An Architecture for Supporting Multi-thread Dialogue

Usually, in a world model UI, there are several items such as notepad or memo, with which a user can choose to have dialogue. Each item that a user interacts with corresponds to an Interactive Object in the UI. Dialogue switching can be preformed among these Interactive Objects. As each Interactive Object is a self contained entity, it has its own dialogue controls and can remember the user input sequences. Hence, its dialogue can be temporary suspended and then resumes later by recovering the previous state of the dialogue. Users can have dialogue with several Interactive Objects independently.

If a user has chosen a particular Interactive Object, we say that the chosen Interactive Object is activated. Subsequently, all incoming messages from the user will be sent to this activated Interactive Object. An interaction knowledge base in a UI contains the rules, constraints and current global states for each Interactive Object in the UI. Therefore, it can monitor which Interactive Object should become activated or de-activated. It directs all incoming messages from the users to the activated Interactive Object according to its current states. The

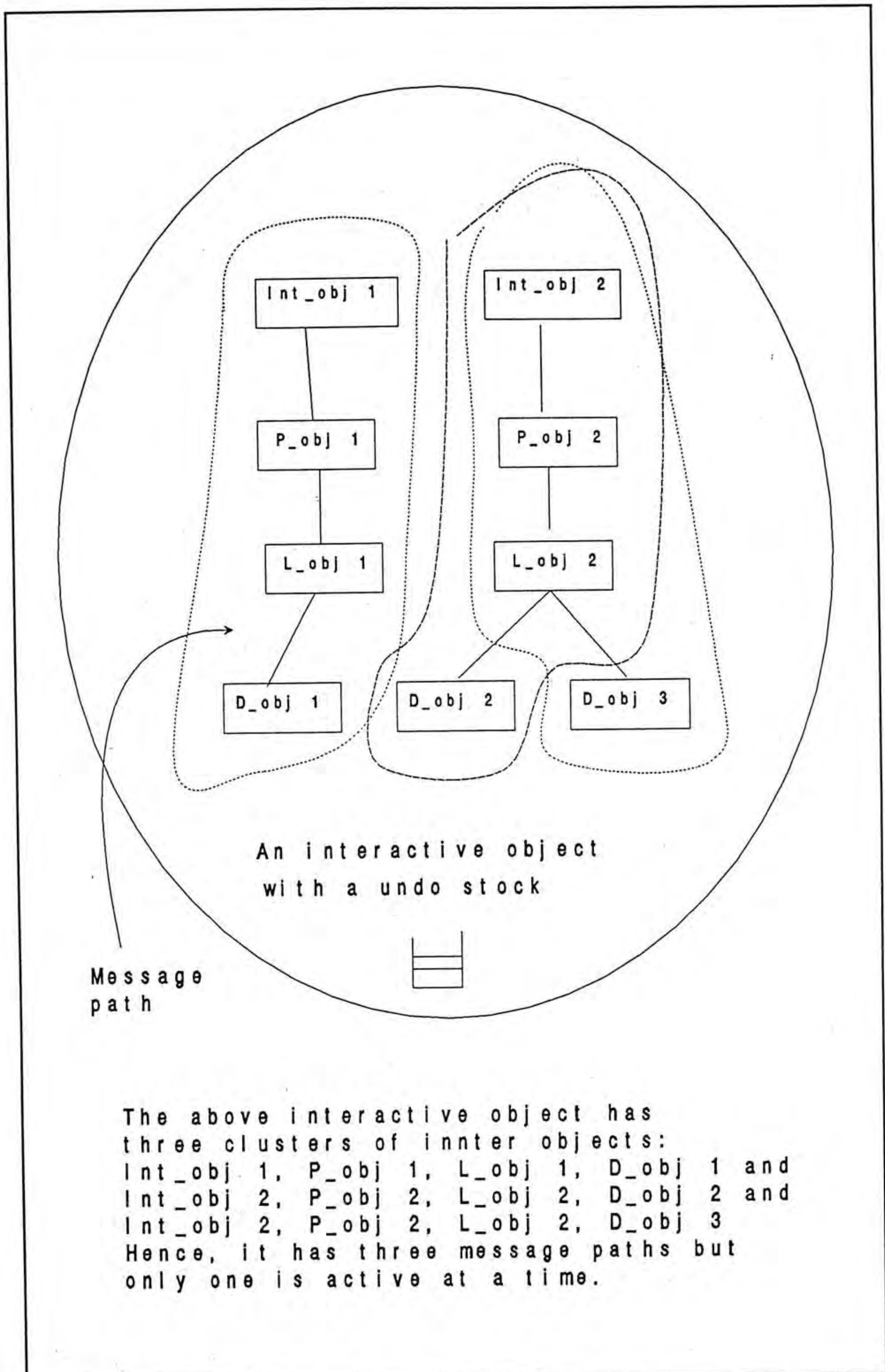


Figure 3.12 A skeleton of interactive object

interaction knowledge base can be considered as an extension of the cluster controller in an Interactive Object. However, unlike the cluster controller in an Interactive Object, interaction knowledge base connects the message path for activated Interactive Objects rather than the inner objects within an Interactive Object. Only the activated Interaction Object in the activated Interactive Object can interact with the interaction knowledge base. The dialogue switching can be facilitated by a switch box mechanism to be discussed in section 4.1.4.

3.4 A Basic Structure of an Object

In this section, a new design specification is proposed to specify an object in a UI. This design specification notation also specifies dialogue control in an object.

Each object in the Object-Oriented UI Model, including undo object, Interactive Object and its inner object, belongs to a class and has the following template:

#1 **OBJECT** {object_name} **IS SUBCLASS OF** { list of base classes }

#2 **INCOMING MESSAGE**

#3 { messages that can be handled by this object }

#4 **OUTGOING MESSAGE AND ITS DESTINATION OBJECT TYPE**

#5 { messages that will be send out and its destination object type }

#6 **METHOD**

#7 { functions used in this object }

#8 **INSTANCE VARIABLE**

#9 { instance variables used in this object }

#10 **DIALOGUE SPECIFICATION**

#11 { dialogue control described by event language notation }

Line 1 specifies the name of an object and its base classes. Lines 2 and 3 specify the incoming messages that this object will handle. Lines 4 and 5 specify the outgoing messages and its destination object type. It is worth noting that only the destination object type is specified rather than any particular instance object itself. That is the object does not know the exact destination of the outgoing messages but only a class of objects that the messages may be sent to. This will be discussed more in section 3.4.5. Lines 6 to 9 specify the functions and instance variables used in this object and both of them can be inherited by its subclasses. The implementation of the functions should be hidden from outside objects. Lines 10 and 11 specify the dialogue control which can be modeled by an event model [23,29].

3.4.1 An Event Notation for Dialogue Control

In the event notation, a dialogue control is monitored by an Event Handler (EH) which is described by a Event - Response Language (ERL) [23,29,32,37]. The main elements of ERL are incoming events, outgoing events and flags. An event is a signal that something has occurred and it may carry data too. Event can be considered as a kind of message that passed from one object (may be input device) to another. Flags are variables used to encode the state of the dialogue and to control execution in the event handler. Each incoming event has a rule associated with it. A rule consists of two parts: condition and action. It has the

following form:

condition -> action

Condition can be a list of flags and/or an incoming event. Action can be assignment statements (including raising flags), procedure calls and/or sending outgoing messages to other object. Rules whose condition does not contain any event are called ϵ-rules; otherwise called regular rule. ϵ-rules are used for consistency checking and execution control in the event handler. The action in a regular rule can be executed only when

1) the incoming event is at the head of an event queue and

2) all flags in the condition are raised i.e the condition is "open"

As for ϵ-rules, their actions will be executed only when their conditions are open. Flags in a condition can be considered as guards to trigger action in a rule. Regular rules for the incoming messages and ϵ-rules are grouped together to form an Event Handler which is used to monitor the dialogue control in an object.

Regular rules are evaluated only one time per incoming message. But ϵ-rules can be evaluated more than one time as long as their conditions are open. The evaluation of ϵ-rules are repeated until all the conditions in the ϵ-rules are closed. Then the object is said to be stable.

In the Event Handler, there is usually a ϵ-rule to send an incoming message to its upper abstract level object as mentioned in section 3.3.7. The flags in the condition of this ϵ-rule will raise if the incoming message do not fall in the incoming message list specified in lines 2 and 3. As mentioned in section 3.3.6, any incoming message that cannot be handled by the object should be passed to

its upper abstract level object. For example, as shown in Figure 3.9, if a Lexical Object cannot handle the incoming message then the message should be passed to a Presentation Object which is its upper abstract level object.

It has been shown that [23,29,32] event model is suitable for developing graphical UI. However, we are difficult to capture user input sequences from event notation; hence, it is not suitable for early dialogue specification. Nevertheless, there is an algorithm [29] to convert the transition network notation, which is more understood by human, to event notation. For more details about this algorithm, readers can refer to Appendix A1.

3.4.2 Maintaining Consistency through ϵ-rules

Unlike traditional conversational UI, model world UI always allows a user to view the current status of an Interactive Object all the time. That is the UI may have to update its output to make the output to be consistent whenever there is a state change in UI. For example, after we have preformed a "cut" command, the UI should enable the "paste" command item in the command menu to allow the user choose this command. This updating operation should be done automatically regardless if there is a user input.

As mentioned in section 3.4.1, the condition part in ϵ-rules does not involve any incoming event. The action of an ϵ-rule is fired automatically as long as its condition is open. When a user inputs a command and changes the state of an Interactive Object, some actions may need to be taken to maintain the consistency of the system. In such a case, some flags will be raised after processing the input command so that the actions of ϵ-rules which contain some

updating operations for maintaining consistency of the system can be fired. All ϵ-rules in an object are evaluated repeatedly until all the conditions in the ϵ-rules are closed. Then all updating operations should have been done and the object is said to be in stable state.

Although most updating operations follow some actions in regular rules directly, we cannot just append the updating operations to the actions in regular rules and eliminate the ϵ-rules. Because several different changes of state may require the same updating operation. For instance, in the above example, the same updating operation should also be preformed for a "copy" command. Therefore, it is better for us to factor out the common updating operations and to group them into ϵ-rules. In addition, ϵ-rules can also provide us a clear and easy understand mechanism for maintaining consistency of a system. The mechanism can be understood in the following ways:

1) a user inputs a command,
2) changes the state of the system,
3) disturbs the balance of the system.
4) the system becomes unstable, and some flags are raised,
5) ϵ-rules are evaluated and at the same time performs updating operations.
6) repeat step 5 until the system becomes stable (i.e. All conditions in the ϵ-rules are closed)

Dialogue switching between Interactive Objects also causes consistency problem. If two Interactive Objects share the same window and when there is a dialogue switching from one to another, the window has to be updated so as to make the display window consistent with current states of the activated Interactive

Object. As interaction knowledge base monitors dialogue switching, it has the responsibility to raise flags to fire actions for updating window.

3.4.3 An Example of an Inner Object Specification

The following object specification is based on the transition network diagram of an Interaction Object shown in figure 3.11.

```
#1    OBJECT Sale_Int IS SUBCLASS OF Interaction Object
#2    INCOMING MESSAGE
#3          copy, cut, clear_clipboard, paste, sum, avg,
#4          DATA_INPUT
#5    OUTGOING MESSAGE AND ITS DESTINATION OBJECT TYPE
#6          <sum,Lnk_obj>, <avg,Lnk_obj>
#7    METHODS
#8          EnableMenuItem(command type);
#9          DisableMenuItem(command type);
#10         ClearClip();
#11         CopyCliptoBuffer();
#12         copyBuffertoClip();
#13         ClearBuffer();
#14         UpdateData();
#15   INSTANCE VARIABLE
#16         Boolean State1, State2, State3, State4,
#17   DIALOGUE SPECIFICATION
#18
#19         DATA_INPUT State1 ->
#20                     UpdateData();
#21                     State2  ↑
#22
#23         DATA_INPUT State2 ->
#24                     UpdateData();
#25
#26         DATA_INPUT State3 ->
#27                     UpdateData();
#28
#29         copy State2 ->
#30                     CopyBuffertoClip();
#31                     State3  ↑
#32
#33         cut State2 ->
```

```
#34                     CopyCliptoBuffer();
#35                     ClearBuffer();
#36                     State3  ↑
#37
#38         Clear_Clipboard State3 ->
#39                     ClearClip();
#40                     State2  ↑
#41
#42         paste State3 ->
#43                     copyBuffertoClip();
#44
#45         sum State2 ->
#46                     <sum,Lnk_obj>!
#47                     State4  ↑
#48
#49         avg State2 ->
#50                     <avg,Lnk_obj>!
#51                     State4  ↑
#52
#53         sum State3 ->
#54                     <Sum,Lnk_obj>!
#55                     State4  ↑
#56
#57         avg State3 ->
#58                     <avg,Lnk_obj>!
#59                     State4  ↑
#60
#61         State2 ->
#62                     DisableMenuItem(Paste);
#63
#64         State3 ->
#65                     EnableMenuItem(Paste);
#66
```

Note, ↑ denotes flag raising while ! denotes message sending operator.

Except rules at line 61 to 65 are ϵ-rules, all the above rules are regular rules. The two ϵ-rules are for menu items consistency maintenance. When the condition of a ϵ-rule is open, its action will be evaluated and then all the flags in

the condition is lowered in order to prevent re-evaluation of the same ϵ-rule.

The message in upper case (DATA_INPUT) corresponds to dialogue control in other object. DATA_INPUT corresponds to the dialogue control in a Presentation Object as shown in figure 3.11. When the Sale_Int object receives the "sum" or "avg" message, it sends the message to its linkage object. Through the linkage object, the Sale_Int object requires the semantic services of "sum" and "average" in the application.

3.4.4 Pre and Post Conditions of Action

Before we write down rules for the Event Handler, the Pre and Post conditions can be used to specify each rule. A Pre-condition is the condition that must be held before an action can be executed; while post-condition is the condition that must be satisfied after the execution of the action. Pre-condition of the action will just become the condition in the rule; while the post-condition is the side effect of the action.

3.4.5 Automatic Message Routing

We have mentioned that when an object sends outgoing messages to an other object, the object itself does not know the exact object to which the messages will be sent. It only knows the type of object that the message may be sent to. Actually, the routing of the outgoing message depends on the active message path in the activated Interactive Object described in section 3.3.10. In turn the active message path depends on the current state of the activated Interactive Object and current global states of the UI. Message routing is totally

hidden from the message sending object. As the interaction knowledge base contains all current states of UI including the sates in the activated Interactive Object, it actually determines how the messages are routed to their destinations. In short, UI routes messages to their destinations according to the current states in the interaction knowledge base.

Automatic message routing mechanism gives great contribution to flexible UI modification and rapid prototyping. As objects need not care about their outgoing message destinations, they can be easily reused in other UIs. In the other way around, a UI can easily replace objects without rewriting the whole program. In the Object-Oriented UI Model, the automatic message routing is used to encourage Software-IC construction which will be described in section 4.1.5.

This automatic message routing mechanism can be implemented by dynamic binding feature which is one of the powerful feature of object oriented programming and will be described in detail in section 5.1.1.

3.5 Systematic Approach to UI Specification

In order to support a systematic approach to UI specification based on our model described above, the following steps are proposed.

1) Identify Interactive Objects

First we should figure out how many Interactive Objects a user may manipulate.

2) Identify interaction style for each Interactive Object

Figure out the possible interaction between a user and Interactive Objects.

Then we separate this interaction operations from application functions.

3) Identify inner objects within each Interactive Object.

Figure out clusters of inner objects within each Interactive Object based on its interaction style.

4) Specify the cluster controller for each Interactive Object.

Specify the active message path for each cluster of inner objects at different states of an Interactive Object.

5) Specify each inner objects in an Interactive Object.

- specify the dialogue control either by a transition network or an event response language
- specify consistency checking for each change of state
- identify incoming message
- identify outgoing message and its destination object type.
- specify methods and variables in the object

6) Specify each rule in an event handler.

- identify regular rules and ϵ-rules in event handler according to the dialogue control and consistency checking specified in the object.
- specify the pre and pre conditions for each rule

7) Specify interaction knowledge base.

- identify global states for dialogue switching and messages routing
- specify functions for updating current global states

Finally iterate steps 3 to 7 until all Interactive Objects satisfy the original user requirement.

Chapter 4

User Interface Framework Design

This chapter describes a user interface framework design for the Object-Oriented UI Model described in chapter 3. The UI framework is a basic foundation for UI development. Any model world style UI can be developed from this framework.

4.1 A Framework for UI Development

Our UI framework can be considered as a basic foundation for UI development. It contains some null methods and default values and can be turned into a self-contained, complete UI. However, if we only use the UI framework and do not extend the framework by overriding the null methods and default values in the framework, the UI produced from this UI framework will do nothing for us. The purpose of this UI framework is just to provide a basic blue print for UI development. We start from this UI framework and extend it so that this extended UI framework can generate a complete UI design that satisfies our original objectives. The extension of UI framework should be easy and efficient so that UI designers can develop their UIs quickly from the existing UI framework. Fast development of UI from UI framework also encourages rapid evolutionary prototyping in UI development life cycle.

We first look at the basic structure of the UI framework and then discuss its implementation in the Microsoft window 3.0 and C++ 2.0 environment in Chapter 5.

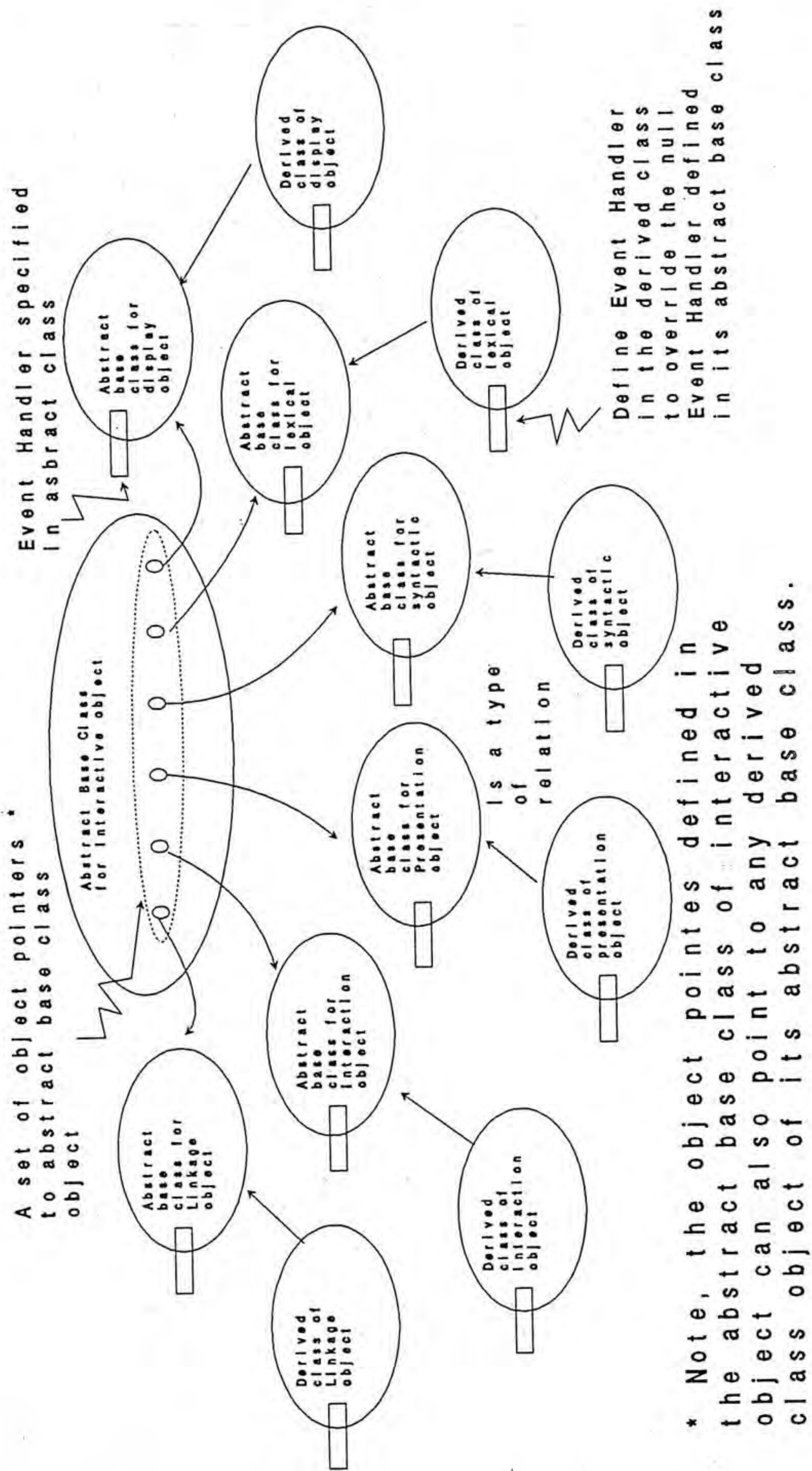


Figure 4.1 Define abstract base class for each object

4.1.1 Abstract Base Class for Each Object Type

As the design of UI framework is based on the Object-Oriented UI Model described in Chapter 3, the UI framework should also have such basic objects as interaction objects, presentation objects, display objects, specified in the Object-Oriented UI Model. A set of abstract base classes of these basic objects is defined in the UI framework as shown in Figure 4.1. The UI framework has the abstract base classes for interactive object, linkage object, interaction object, presentation object, syntactic object, lexical object and display object. The objectives of an abstract base class is to provide basic construction and standard interface part for each object in the UI. Abstract base classes can contain some default values and null methods. Default values in an abstract base class are only used if the derived classes of the abstract base class do not override them. Null methods in an abstract base class are usually overridden by the derived classes of the abstract base class as these null methods do not perform anything at all. The purpose of these null methods is to provide standard interfaces for derived class objects during dynamic binding. The feature of dynamic binding in C++ Object-Oriented programming will be described in section 5.1.1.

The reason for overriding the methods in the abstract base classes is to utilize the polymorphism feature in Object-Oriented paradigm. These abstract base classes only provide a basic UI skeleton for UI development. In order to develop a UI that is customized to our users' needs, we have to extend these abstract base classes. One of the methods to extend these abstract base classes is to override these abstract base classes by their derived classes. When we want to modify some properties of the UI framework, we do not need to modify the

objects in the abstract base classes. Instead we only need to add derived objects to override objects in the abstract base classes as shown in Figure 4.2. By introducing proper derived objects into the UI framework, we can keep the basic UI framework intact as we extend the UI framework for UI development.

The null methods in the abstract base classes also provide standard interface part for the derived classes of the abstract base classes. Any derived object which overrides the null methods in their abstract base classes must have an interface part specified in the null methods in their abstract base classes. For examples in Figure 4.3, in order to override the method MA2 in the abstract base class object A, MB1 method and MC1 method in the derived object B and C must has the same interface as MA2 method in the abstract base class object A. Figure 4.3 also points out that a derived object can reuse any methods in its base objects providing that it does not override these methods. By overriding and reusing the properties of the base objects, we can shape the derived objects in different ways so as to satisfy our design needs.

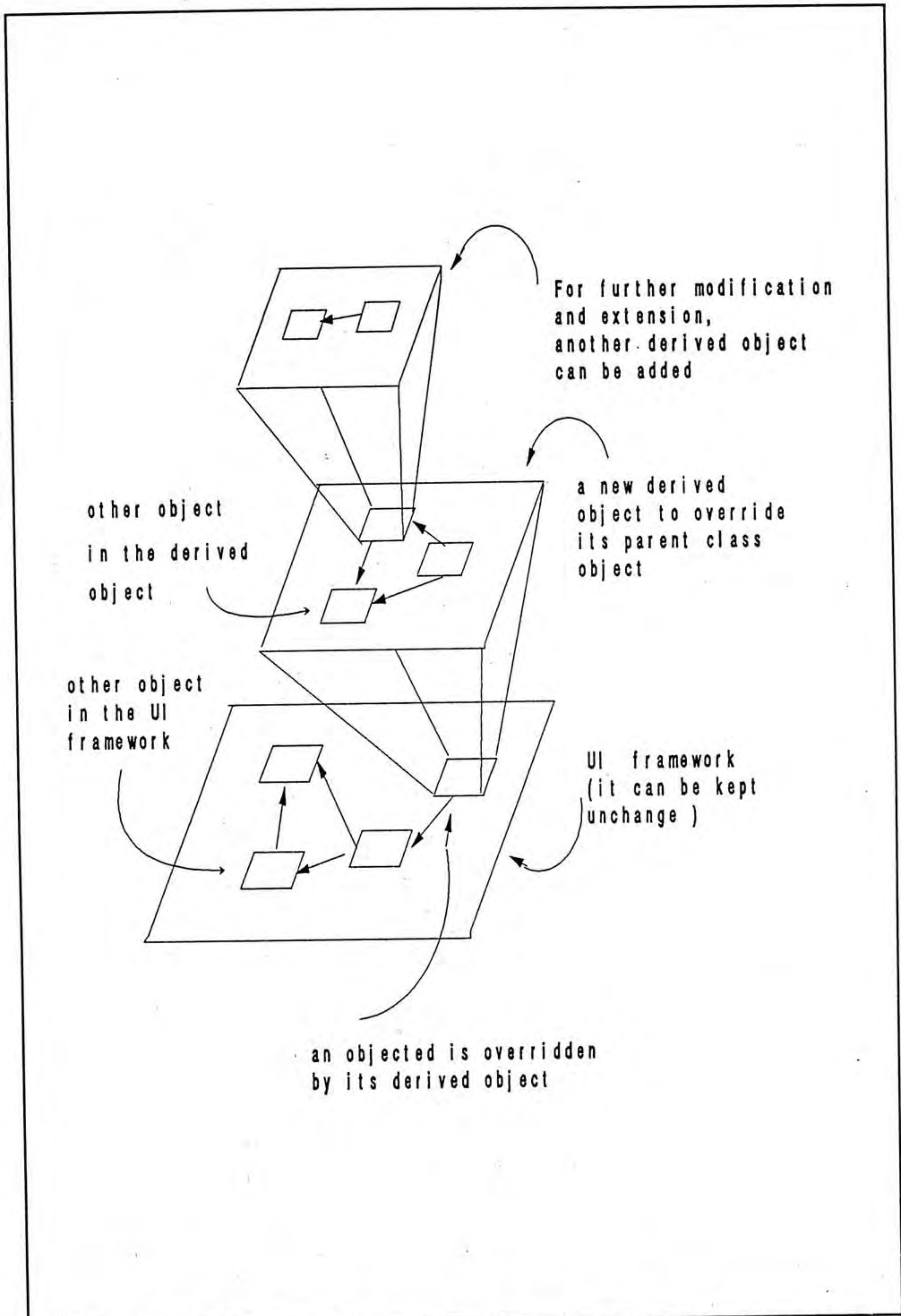


Figure 4.2 Derived objects override objects in the abstract base class.

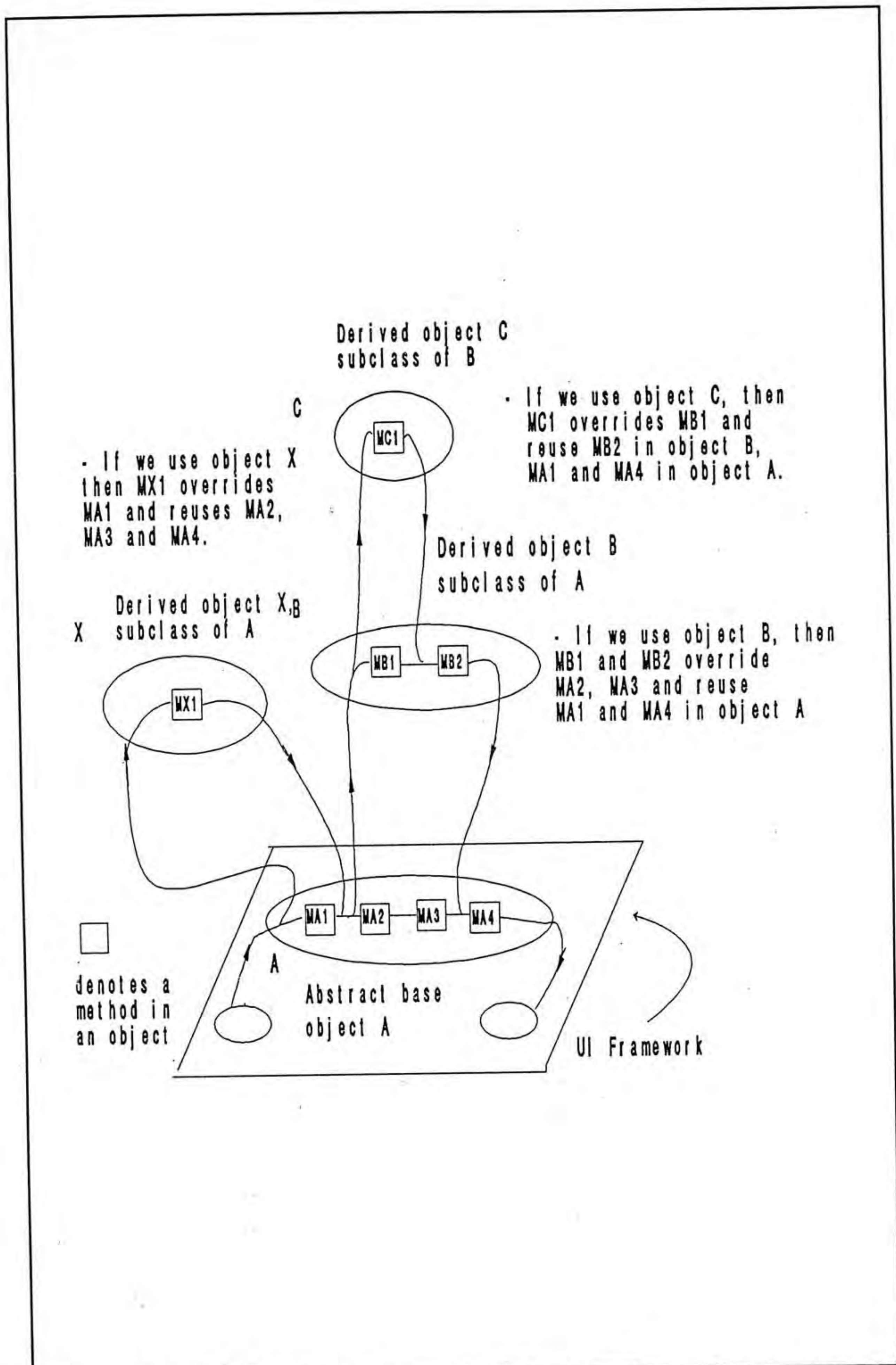


Figure 4.3 The basic methods in the abstract base classes provide standard interface for their derived classes.

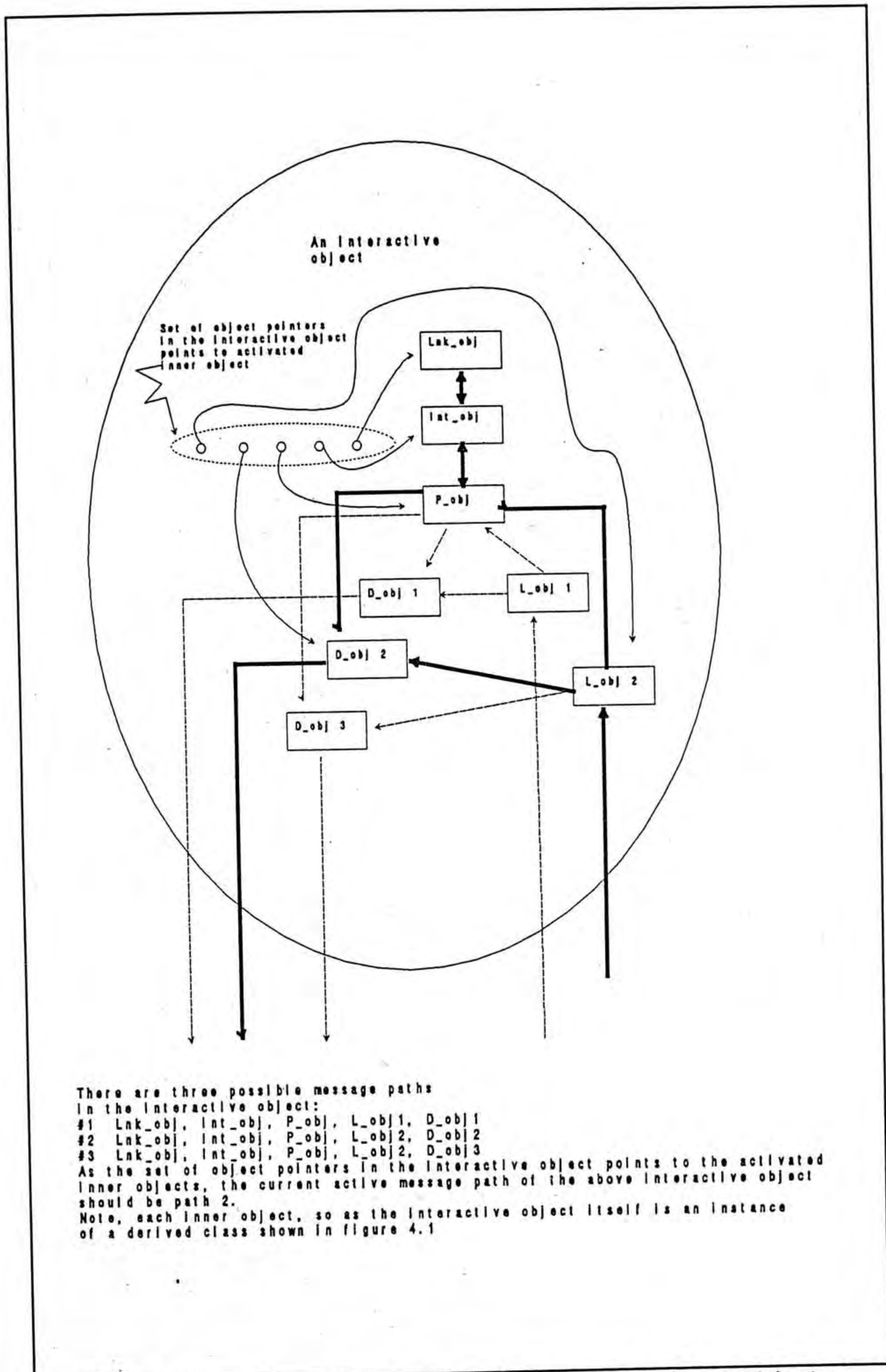


Figure 4.4 Object pointers point to the activated inner objects

Each abstract base class at least has a null event handler which is supposed to be overridden by its derived class as shown in Figure 4.1. As all incoming messages will go to an event handler first, the null event handler in the abstract base class provides a standard message communication interface for all derived class objects.

In order to encourage code sharing, we extract all the common properties of the derived classes into their abstract base classes and let these common properties be inherited by their derived classes. For example, in an interactive objects, there is a set of activated inner objects (linkage object, interaction object, presentation object, syntactic object, lexical object, display object) at all the time no matter what type of the interactive object it will be and how many inner objects it will have. Hence, in the abstract base class of interactive object, there is a set of abstract base class object pointers which is used to point to the activated inner objects as shown in figure 4.4. This set of pointers, which are declared in the abstract base class of interactive object, are used to identify the active message path in an interactive object and can be inherited by all the derived interactive objects.

Abstract base classes provide an environment for object polymorphism and inheritance which facilitate easy modification of objects. As a derived object can inherit all properties of its base class object, the derived object can reuse all the methods and variables in its base class object. Hence, inheritance encourages reusability and sharing of codes. It also eliminates repetitive coding which usually occurs in most UIs. Polymorphism helps us to modify software components more easily. If we want to modify an object, we do not need to rewrite the object.

Instead, we define a derived class of that object and use this derived object to override some of the original object methods and attributes so as to achieve our modification. And at the same time this derived object can reuse its based class methods and attributes that have not been overridden through inheritance as shown in Figure 4.3. Easy modification of object can encourage rapid prototyping for UI development which is also very important for UI development life cycle.

4.1.2 A Kernel for Message Routing

The framework also includes a kernel for handling message passing among objects in a UI. All message passing among objects is performed through this kernel. Message package is first sent to the kernel and then the kernel enques it at the tail of an event queue. While the event queue is not empty, the kernel deques a message package from the head of the event queue and sends the message to its destination according to the message type and the current state of the UI. The kernel as shown in Figure 4.5 contains the event queue which holds message packages received from objects. A message package contains the message itself and a message type. The message type specifies the type of object which will receive this message. The content of the message is totally transparent to the kernel. The message type can be obtained from the message package; while the current state of the UI can be obtained from the Interaction Knowledge Base. It is worth noting that the message type only specifies the type of object and not any object instance. For example, in an UI there are several interactive objects and each interactive object may contain several display objects. The message type may specify that the message will send to an object belonging to a display object class

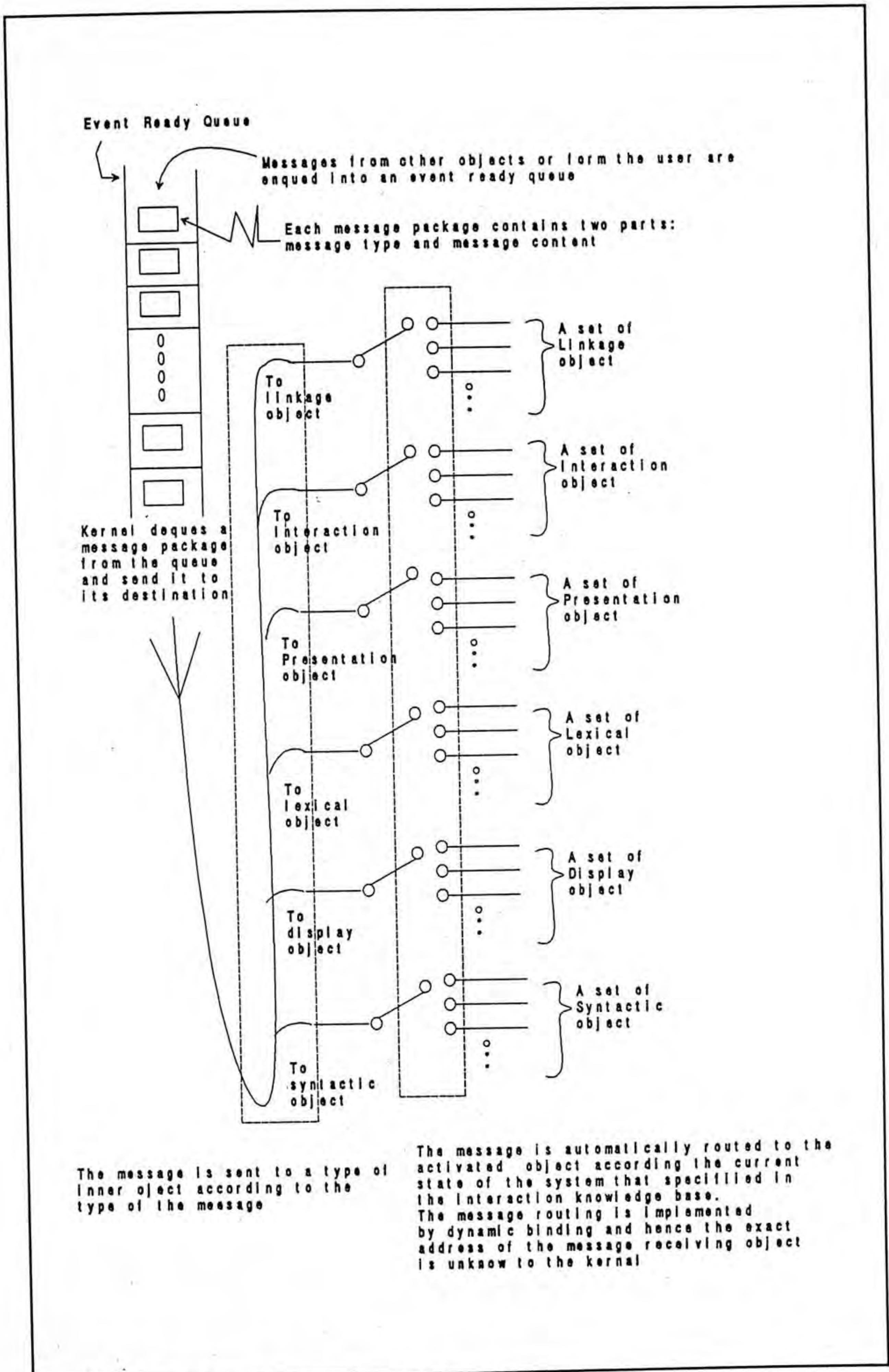


Figure 4.5 A kernel for message passing

but it does not specify any display object to receive this message.

The global current states of the UI are stored in the Interaction Knowledge Base. They include a set of abstract base class object pointers. However, object pointer to base class object is compatible to object pointer to its derived class object. As each object in the UI, no matter it is an interactive object or inner object within an interactive, is a derived object of its abstract base class, the set of abstract base class object pointers in the Interaction Knowledge Base can also point to the activated interactive object and the activated inner objects within that activated interactive object. The kernel, therefore, can send messages to their destinations through this set of object pointers even though the destinations is dynamic and determined at run time.

Actually, the kernel in the framework acts as a listener object shown in Figure 3.8. Its duty is to deliver incoming messages to appropriate objects.

As the kernel only uses abstract base class object pointers to send message and the type of abstract base class object pointers is compatible with all their derived classes (even though the types of object pointed by this pointers may vary at run time), the code of kernel can be kept unchanged no matter what kind of derived classes have been introduced in the UI. Actually, the types of object pointed by this abstract base class object pointers are transparent to the kernel. Hence, no matter how many new types of object have been added into the UI, the kernel is unaffected.

In the other way around, this kind of message routing mechanism can also release the message sending objects from the inconvenience of deducing the exact address of message receiving objects. This deduction usually requires knowledge

of global UI configuration. Under this message routing environment, the message sending objects only have an intention to send a message to a certain class of objects and need not know the exact address of the message receiving objects. The addressing of the message receiving objects should be taken care by the kernel and the Interaction Knowledge Base. Consequently, when we specify a new interactive object and its inner objects, we can ignore global configuration of the UI. Such information hiding of the global configuration of UI encourages the construction of software IC in the UI development process which will be discussed in 4.1.5.

4.1.3 Interaction Knowledge Base

The Interaction Knowledge Base has a reservoir of pointers to all existing interactive object pointers as shown in Figure 4.6. It monitors which interactive object is activated or de-activated based on the current state of UI. As different UIs may have different number and/or different types of interactive objects, Interaction Knowledge Base varies from UI to UI. Therefore, no universal Interaction Knowledge Base for all UIs exists. Whenever interactive objects are added or removed from an UI, the Interaction Knowledge Base has to be updated. However, this updating can be easily accomplished by just adding or removing object pointers. In fact, the interactive objects can be even dynamically created or deleted through the object pointers at run time.

The Interaction Knowledge Base can also be considered as a global controller for the interactive objects. Without it we cannot develop a UI from the UI framework. In order to make the UI framework self-contained and complete,

香港中文大學圖書館藏書

a minimum Interaction Knowledge Base should be maintained in the framework.

Through its object pointers, an Interaction Knowledge Base can provide address for the kernel to deliver messages. Besides the pointer to activated interactive object, it also supplies the pointers that point to the activated inner objects within the activated interactive object to the kernel for message delivery but it does not deduct the pointers to activated inner objects. Actually, it gets them directly from the activated interactive object. As it has mentioned in section 4.1.1 above, within an interactive object, there is a set of object pointers which point to the activated inner objects within each interactive object. The updating of these object pointers within the interactive object is done by the interactive object itself. Actually, the configuration of the activated inner objects should be hidden from both Interaction Knowledge Base and kernel. That is the structure of Interaction Knowledge Base and kernel can be kept unchanged, no matter what configuration of the activated inner objects is. The Interaction Knowledge Base only supplies the address of the activated inner objects to the kernel through this object pointers declared in the interactive object.

4.1.4 A Dynamic View of UI Objects

In a UI, all interactive objects and their inner objects are created when the Interaction Knowledge Base comes into existence. Although all interactive objects are created at the very beginning, only one interactive object becomes activated at a time. The initial activated interactive object can be set by a default object pointer in the Interaction Knowledge Base. Once the activated interactive object has been set up, all incoming messages from a user will be automatically routed

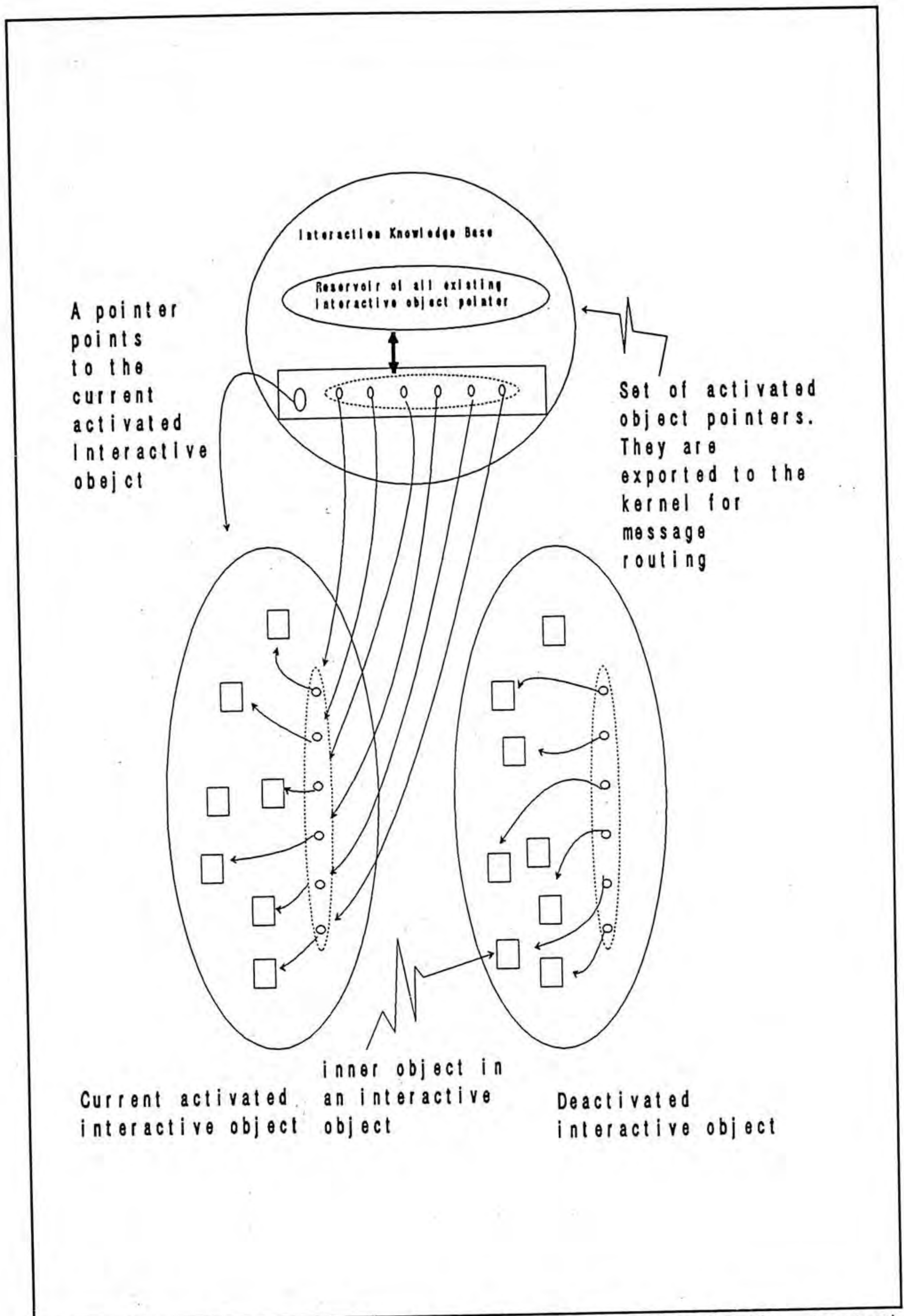


Figure 4.6 Interaction knowledge base keeps track the current activated interactive object

to this activated interactive object.

Actually, interactive objects or its inner objects can be dynamically created or deleted at run time. Objects are created when they are allocated by new object pointers in the Interaction Knowledge Base. Objects are deleted when they are no longer pointed by the object pointers in the Interaction Knowledge Base. An object receives a message when it is activated and the incoming message is a member of its incoming message list which has been described in section 3.4.

An object sends out a message according to its dialogue control. When a rule of the object dialogue specification is triggered by its condition and its action part contains a message sending operator (!), a message is sent to another object. The routing of the message among objects has been described in section 3.4.5.

4.1.5 Switch Box Mechanism for Dialogue Switching

With the help of Interaction Knowledge Base and kernel, dialogue switching between interactive objects can become very simple and easy. As each interactive object is a self-contained entity, it has its own interaction dialogue control and hence has the capability to interact with a user on its own and needs not bother any other objects outside. If a dialogue switching is required from one interactive object to another, all the UI needs to do is to update the activated interactive object pointer in the Interaction Knowledge Base. Then all incoming messages from a user will automatically be forwarded to the new activated interactive object. The message passing among the activated inner objects within the activated interactive object is determined by the activated interactive object itself and there is no need for the Interaction Knowledge Base to take care of the

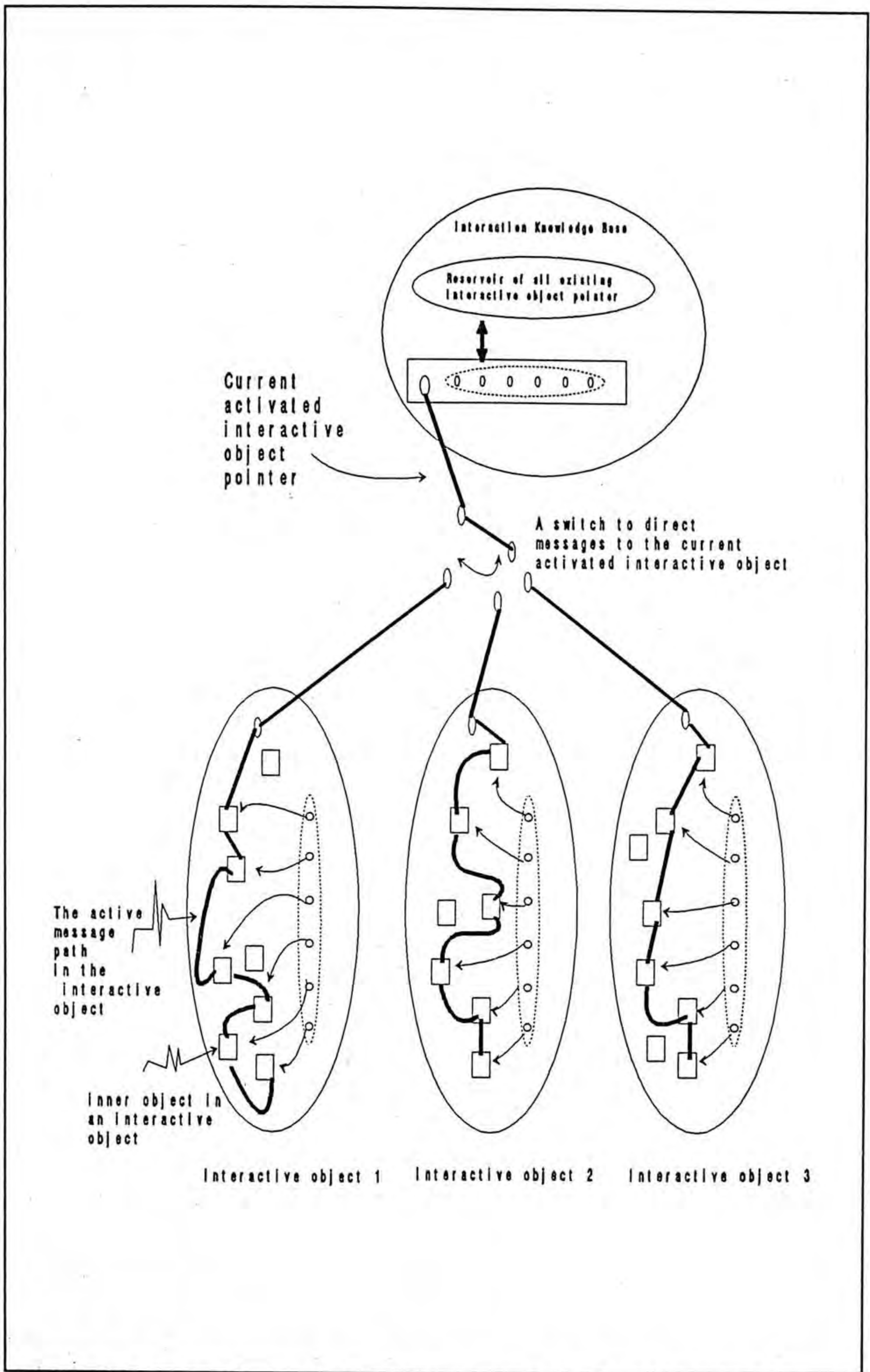


Figure 4.7 A simple switch box mechanism for dialogue switching

message passing among the activated inner objects. Under such an environment, dialogue switching can be viewed as a simple switch box mechanism in which only a single parameter is needed to be updated in order to reconnect all message paths for a new activated interactive object.

4.1.6 Software IC Construction

In order to introduce software IC concepts in our software construction, software components in a system should be considered as self-contained entities and should depend on each other as less as possible so that they can be easily added, removed and exchanged as hardware ICs do. But at the same time they should be integrated together efficiently and can co-operate with each other so as to give an optimal performance. Similar to hardware ICs, standard interfaces for software ICs is needed for communication between objects. Communication between software ICs is done by message passing between them. During communication, the roles of software ICs are considered as client and service rather than caller and callee in a traditional routine call.

In our model, each object is constructed as a self-contained and independent object as each object has its own dialogue control and event handler to process incoming messages. Specification of null event handler method in abstract base classes can provide a standard communication interface for the derived objects of the abstract base classes. As kernel and Interaction Knowledge Base can provide an efficient automatic message routing mechanism for communication between objects, each object can ignore global configuration of the system. Highly independence, standard interface and global configuration

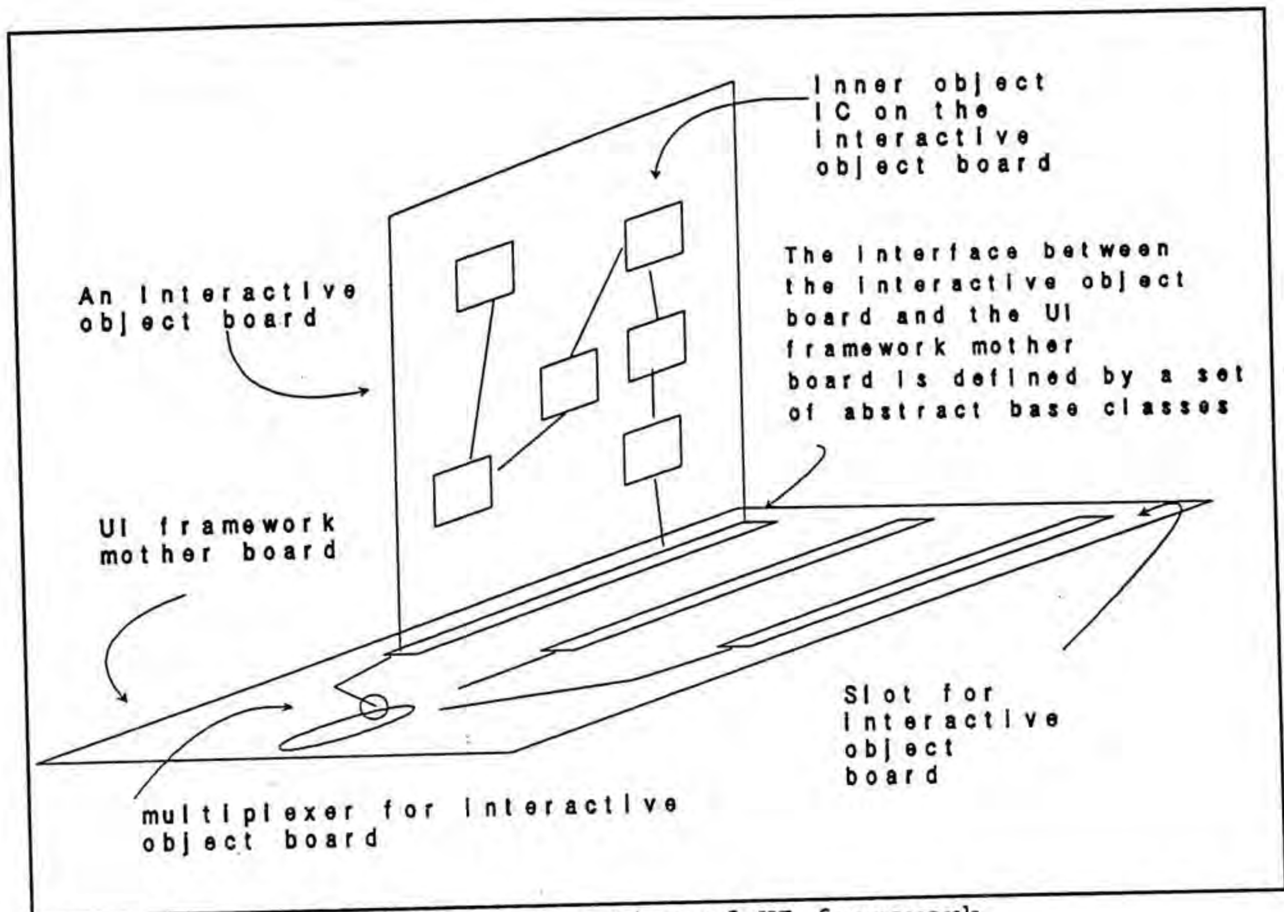


Figure 4.8 Software IC construction of UI framework

ignorance make objects can be added or exchanged flexibly without interfering with other software component. In other way around, because of the above advantages, an objects can be reused by other UIs with different global configurations.

When we develop UIs from UI framework, we can view the framework as a mother board with infinite number of slots. Each slot can hold an interactive object board which can has several inner object ICs on it. The slot interface is specified by the abstract base classes. Dialogue switching is performed as an circuit switching among interactive object boards by a multiplexer in a controller (Interaction Knowledge Base) on the mother board. Each IC has its own circuit (dialogue control) to handle incoming signal (message). All ICs can communicate with each other through a common bus (event queue in the kernel). Chip select of each IC is controlled by a decoder (set of object pointers) based on the current

state of the mother board and the activated interactive object board.

However, there are some differences between the above hardware design analogy and the actually software UI framework design. In hardware design, the chip select control is completely determined by the current state of the system. But in our framework design, beside the current state of UI, message type in a message package also determines message routing between objects.

4.2 **Summaries of Object-Oriented UI Model and UI Framework**

4.2.1 A New Approach to User Interface Development

The following table summaries the differences between traditional UI development and the one based on our Object-Oriented UI Model.

Development approaches based on Object-Oriented UI Model	**Development approaches based on traditional software development**
Realization of specification through object decomposition	Realization of specification through function decomposition.
Multiple dialogue controls which are specified by event response language and distributed among objects	Only one single main dialogue control which is specified by transition network
Control of objects is by message passing which is dynamic, asynchronous and automatic routed	Control of software components is by routine call which is static and sequential
Provides structural design through UI framework	No framework. Design is case oriented and specific to certain application
Information hiding for supporting software IC construction	Information hiding is not enough to support software IC construction

Table 6.1 The contrast of UI development approaches between the traditional UI development and the one based on Object-Oriented UI Model.

4.2.2 Features of UI Development provided by the Object-Oriented UI Model and UI Framework

One of the objectives of the Object-Oriented UI Model and UI Framework is to provide methodologies and simple mechanism for UI development so that the following UI development features can be achieved.

1) Support Separable UI through Linkage component.

The Linkage component in the model acts a mediator between UI and application. It links up the two separate components and provides a channel for them to comminute with each other at high abstraction level so as to achieve dialogue independence.

2) Support Multiple Continuous Feedbacks

The purpose of decomposing an interactive object into several inner objects on different levels is to let each inner object give its feedback at its own level without disturbing other objects so that multiple continuous feedbacks to user at different linguistic levels are possible.

3) Support Multi-thread Dialogue

The model supports the multi-thread dialogue control through a simple switch box mechanism.

4) Support Automatic Message Routing

The message destination can be unknown to a message sending object. That is a message sending object does not know the exact object to which the message will be sent. The message will finally reach the appropriate object according to the current status in the Interaction Knowledge Base.

5) Support UNDO Mechanism

Undo objects in a single generic UNDO stack in each interactive object supports UNDO functions for different operations even at different abstract level.

6) Support Consistency Check Mechanism

Consistency Check can be made by evaluating ϵ-rules in the event handler. The flags in the ϵ-rules can be raised by other rules within the object or by other objects outside through message passing.

7) Support Software IC Construction

Mutual ignorance between objects such as information hiding of object implementation, processing incoming messages by event handler and automatic message routing, makes objects become more independent and hence can be easily interchanged with each other without interfering with other objects. Standard interfaces provided by abstract base class definitions for dynamic binding also facilitates software IC construction.

8) Provide systematic methods to specify and develop UIs

The model provides systematic methods to specify and develop UIs through object decomposition as described in section 3.5. The UI framework also provides a basic standard structure for UI development. The dialogue control in each object can be first specified by a transition network which is more easy to be understood by human and then the transition network is converted into an event response language which is more easy to implement based on the Object-Oriented UI Model.

Chapter 5

Implementation

This chapter presents the implementation of a UI framework whose design has been described in Chapter 4 and a Simple Stock Market Decision Support System (SSMDSS) in the Microsoft Window 3.0 Environment. The SSMDSS is implemented based on the UI Framework. Some desirable features of model world style UIs such as multi-thread dialogue and undo functions which have been stated in Chapter 2 are also illustrated in SSMDSS. The results of the above implementations will be discussed in Chapter 6.

5.1 Implementation of UI Framework in Microsoft Window Environment

The UI framework was implemented according to the UI framework design described in Chapter 4 and was written in Zortech C++ language (Zortech, version 2.0). As the framework was developed in the MicroSoft (MS) Window environment, each UI produced from the framework is a window process scheduled by the window management system in the MS Window environment. The MS Windows Software Development Toolkit (version 2.0) is also used to call Window routines in the MS Windows toolkit library.

5.1.1 Implementation of automatic message routing through dynamic binding

As pointed out in Chapter 4, messages delivery in the kernel is done by dynamic binding through the null event handler interface specified in the abstract base class. Each abstract base class has a null event handler function declared as a virtual function which is used for dynamic binding and is supposed to be

overridden by its derived objects. Actually, in the kernel, all message passing among objects are implemented by an event handler function call of a message receiving object. Therefore, this virtual event handler function in the abstract base class provides a standard interface for message communication between its derived objects. Although the message passing between objects is implemented by an event handler function call, the function call is dynamic (i.e the function binding is determined at run time) rather than static.

The content of the abstract base class object pointer is supplied by the interaction knowledge base and is determined at run time according to the current state of UI. The abstract base class object pointer can point to any object as long as this object belongs to the derived class of the abstract base class. Kernel can call the event handler of a message receiving object which is pointed by the abstract base class object pointer in interaction knowledge base providing that

1) the message receiving object is a derived object of its abstract base class and
2) the message receiving object also has its event handler function to override the one defined in its abstract base class.

For example, the statement

CurIntPtr -> EventHandler(message)

will call the event handler function of an interaction object which is pointed by the current interaction object pointer, CurIntPtr, in the interaction knowledge base. The content of CurIntPtr is dynamic and is determined at run time. Although the type of CurIntPtr is a pointer to the abstract base class of an

interaction object, it contains a pointer to a derived interaction object since the base class object pointer and derived class object pointer are compatible.

Through dynamic binding, the kernel, and the message sending objects, send messages to their destinations without knowing the exact addresses of the message receiving objects in advance because these addresses are determined during run time.

In short, dynamic binding in object oriented programming gives us a powerful mechanism to defer the code binding of a procedure call until at the moment of the call at run time.

5.1.2 A generic message structure

In order to have a flexible message structure and generic event handler interface, message package is defined as two parts: a message type and message content. Message type indicates the kind of object (e.g. Interaction Object or Display Object ..etc.) to receive this message and is declared as an unsigned integer. Message content is declared as a pointer to void which can be casted into the desirable structure when it is passed to an event handler. The actual parameters of the event handler in all abstract base class objects are declared as the type of this message package.

The message package is defined as

```
class msgpgk { /* message package */
        public:
        unsigned msgtype;  /* message type */
        void*    msgcnnt;  /* message contents */
        };
```

5.1.3 A meta class for object communication

In order to have a universal interface for communicating all objects through dynamic binding mechanism a meta general base class is defined and then all objects, including interactive objects, their inner objects and undo objects, belong to subclass objects under this meta general base class. Consequently, the manipulation of objects, including the object communication through event handler function or pushing and popping of undo object in a generic undo stack, can be achieved through this universal interface defined in the meta class.

The meta class is defined as

```
class meta_obj { public:
        virtual long EventHandler(void* message);
        /*  EventHandler(void*) is defined as a universal
            standard interface for all object communication
            */
        };
```

5.1.4 Software component of UI Framework in the MS Window environment

In the MS window environment, the UI framework also includes two additional components: Windowclass object and Generic object. Windowclass object defines all basic default values for a window such as initial size of window, position of window, type of window ... etc. The default values of course can be overridden by its derived object if necessary. The constructor of windowclass object can be overloaded and can have default input parameters. By supplying different sets of input parameters to the windowclass object constructor, different types of window can be created. The Generic object contains all initialization statements that requires a window come into exist. It also includes some basic code to deal with the MS window manager system. The Generic object co-operate

with the windowclass object and form the basic skeleton of a window.

The UI framework in the MS window environment includes:

- WindowClass Object for window default attributes.
- Generic Object for window initialization.
- Kernel for message passing.
- Gen Object for defining all abstract base class
- Interaction Knowledge Base for global control

The above components, except interaction knowledge base, together with queue (for event queue in the kernel) and stack (for undo process) are complied and linked into a C++ library which can be used in future UI development.

5.2 A Simple Stock Market Decision Support System (SSMDSS)

A Simple Stock Market Decision Support System (SSMDSS) was implemented according to the UI Framework in the MS window environment. SSMDSS is used as an example for illustrating the features of the Oriented-Object UI Model and the development methods described in Chapter 3. The application and UI of the SSMDSS are implemented into two separate window programs and communicate with each other through Dynamic Data Exchange (DDE) protocols provided by the MS window environment. The purpose of using two separate programs to implement the SSMDSS is to stimulate a physical separation environment for UI and application as shown in Figure 3.5. The software hierarchy of the SSMDSS is shown in Figure 5.1. The level of abstraction decreases from inner ring to outer ring.

The application of the SSMDSS only provides three simple application

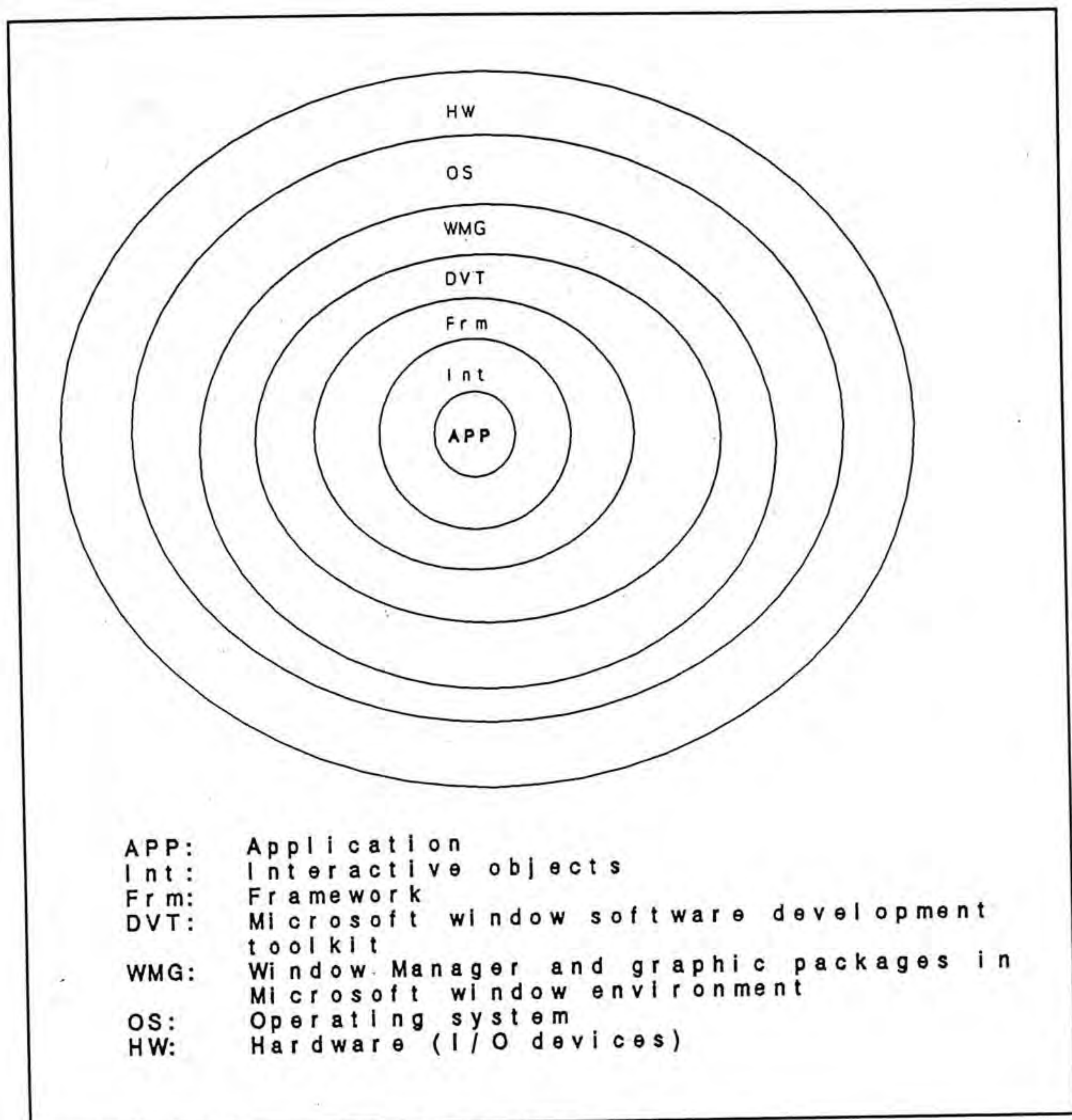


Figure 5.1 Software hierarchy of SSMDSS

functions:

1) - Update the current stock data base,

2) - Retrieve current stock data from the data base and

3) - Predict the trend of stock market according to current stock data and user input.

As we are only interested in the UI part of this system, function 3 was implemented by some pseudo functions so that the detailed algorithms for analyzing stock data are ignored.

In the UI of the system, the following functions are provided.

1 three separate working sheets for three different set of stock data

2 a user can switch to any working sheet at any time as he/she wishes

3 each working sheet can display its stock data either in bar chart, polygon chart or spread sheet form

4 when stock data are displayed in bar chart form, the chart can be viewed by month, by week or by stack

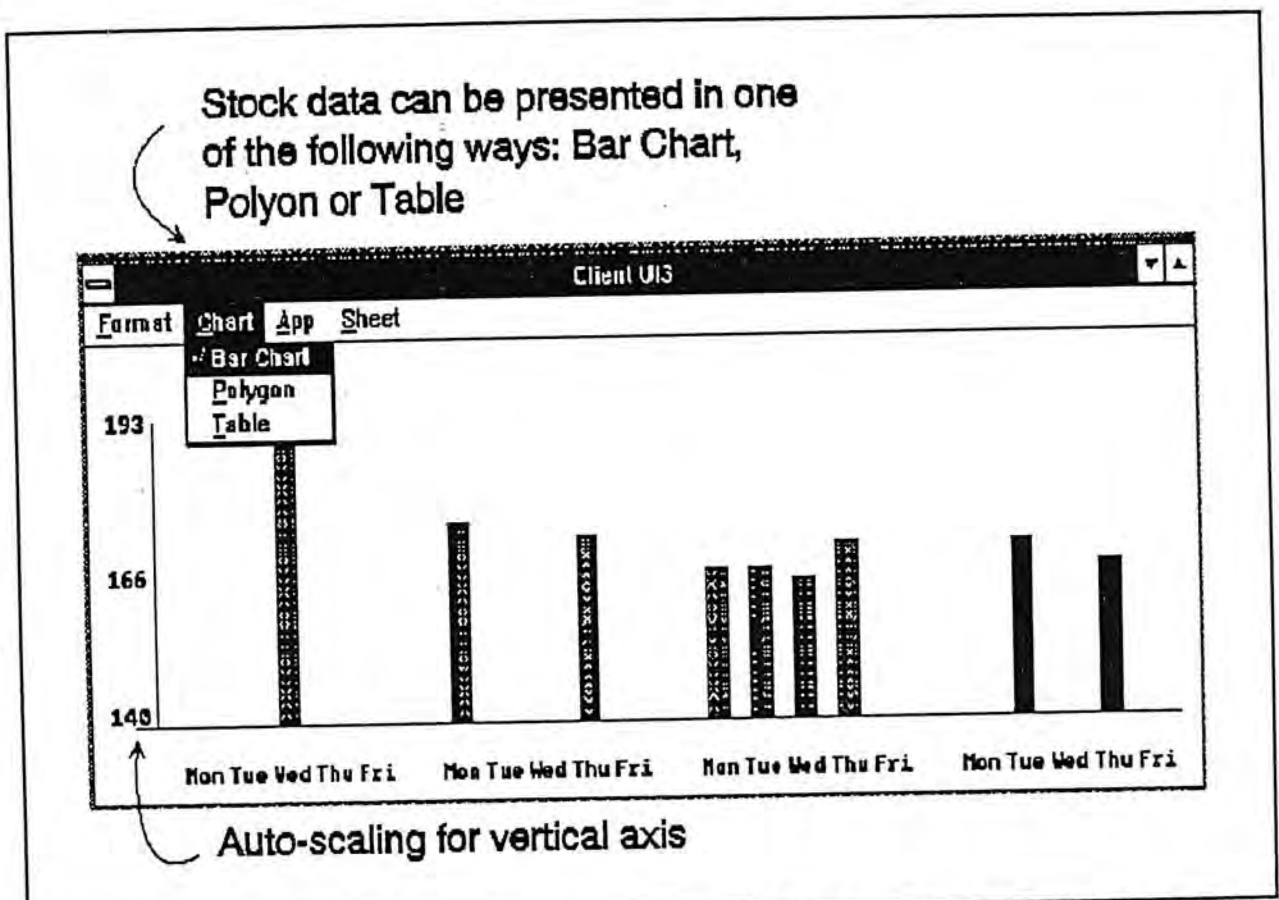


Figure 5.2 The same stock data can be displayed in different ways according to end user choice.

5 when stock data are displayed in polygon chart form, the chart can only be viewed by month or by week.

6 If the chart (bar or polygon) is displayed by week, a scroll bar is provided for a user to scroll the chart week by week.

7 If the chart (bar or polygon) is displayed by week, user can direct

input new stock data by using a mouse drafting on the chart. A stock value which is pointed by a mouse cursor on the chart can be continuously shown to the user.

8 An auto-scaling of vertical axis according to the current stock values is provided.

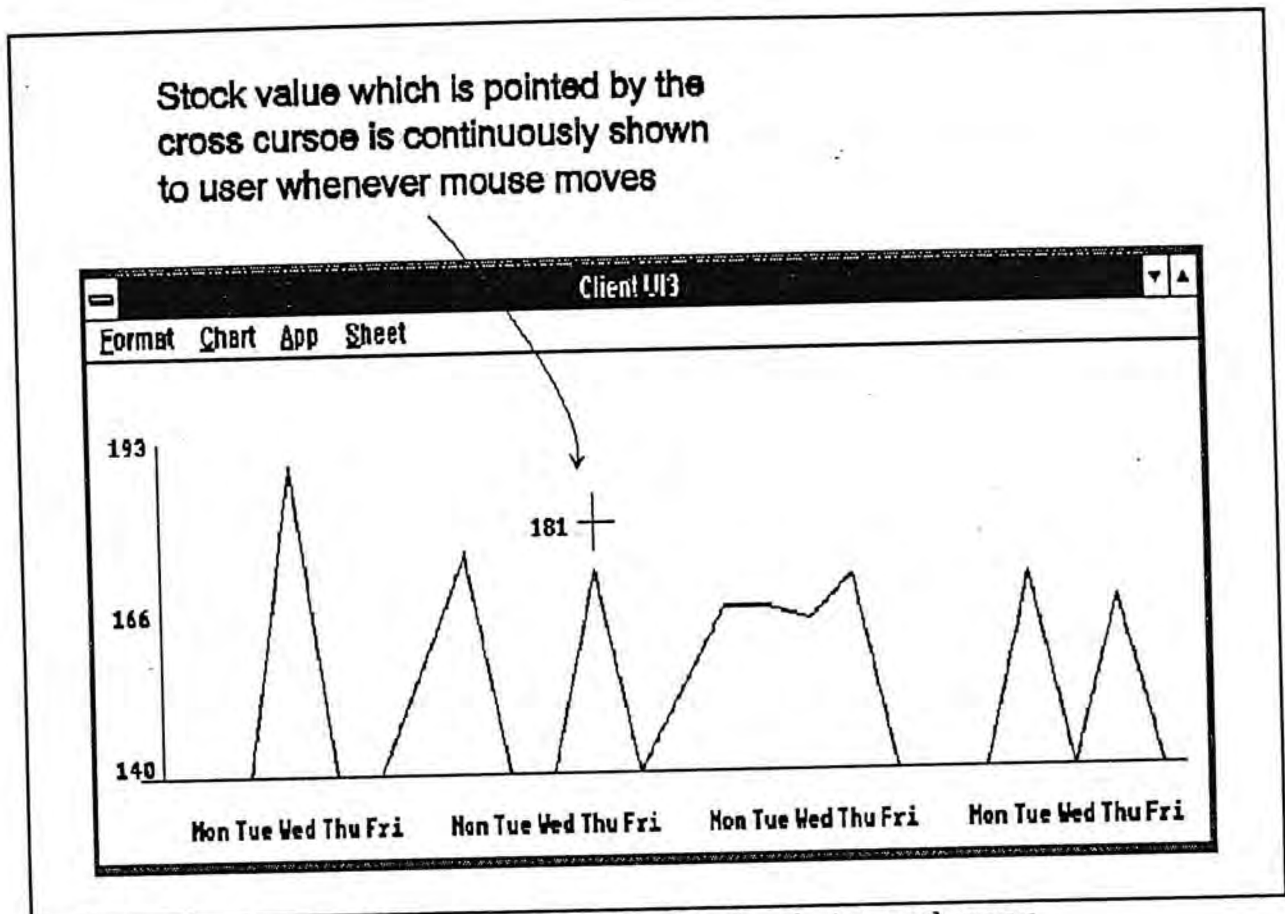


Figure 5.3 A continuous syntactic feedback to end user

9 If stock data are displayed in a spread sheet form, user can choose a stock datum directly either by a mouse or cursor keys on the keyboard and then input the new stock value through the keyboard.

10 The three working sheets can exchange data through copy and paste functions.

11 Provides undo functions for the operations of

- data entry through mouse as the stock data is displayed by week

- data entry through keyboard as the stock data is displayed in spread sheet form
- predicting the trend of stock market
- retrieving stock data from the stock market database
- copy and paste

Undo functions for other operations are not implemented as these operations, except updating stock data in the stock market data base, can be "undo" by redoing other operations provided in the UI.

12 give warning signal if stock value is too low.

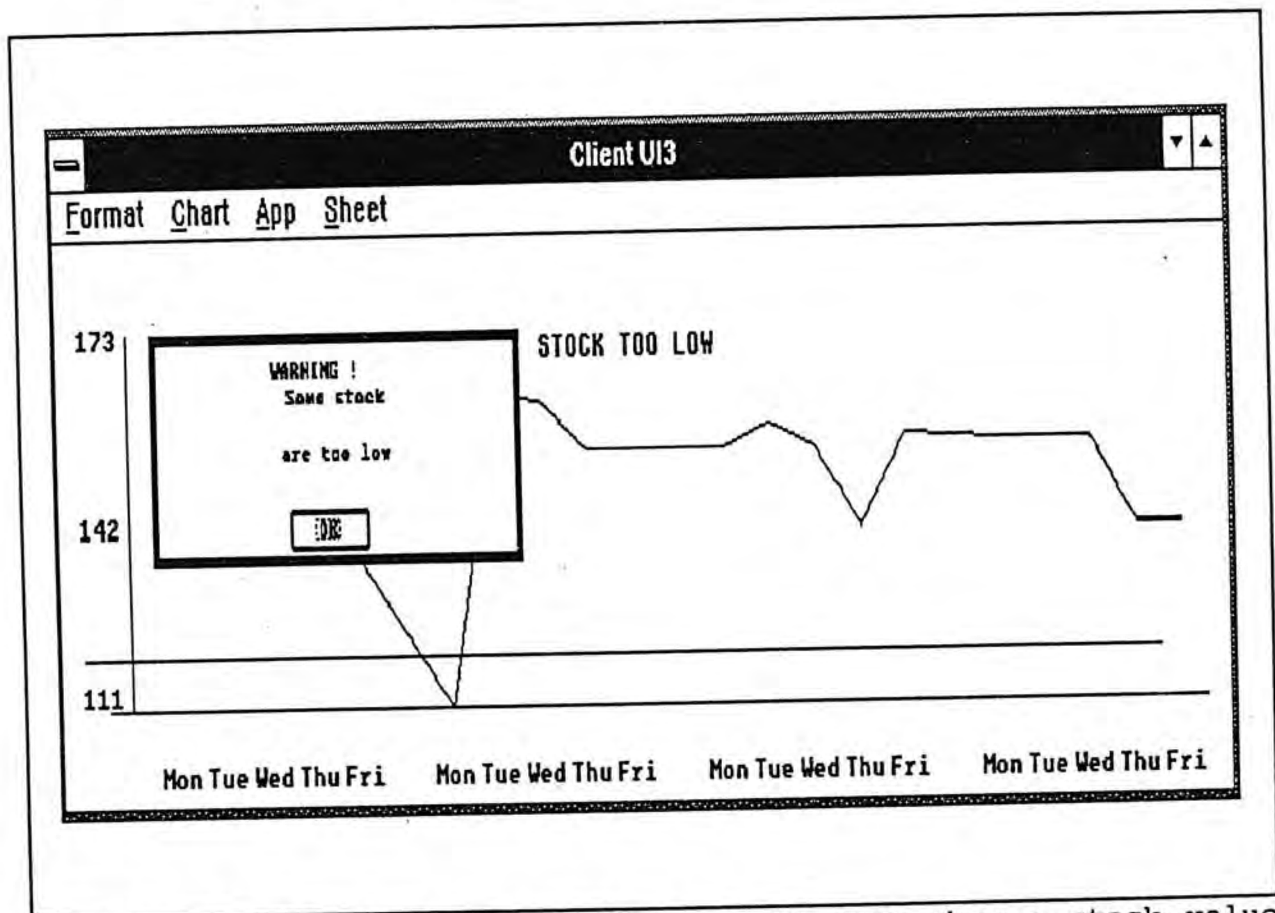


Figure 5.4 A warning message is given to user when a stock value below a safety value

5.2.1 UI Specification

In order to show a systematic approach specification of UI, the UI is specified in the following steps as described in section 3.5.

1) Identify the interactive objects.

As defined in Chapter 3, interactive object is an object with which a user can conduct a meaningful step-by-step dialogue and the user can apply direct manipulation on it. According to the above UI's description, the objects that can be manipulated directly by a user should be the three working sheets for stock data. In order to make the interactive object become more self-contained and complete for dialogue switching, each interactive object has its own undo stack to process its undo objects.

2) Identify interaction style for each interactive object

According to the UI functions described above, the interaction style for each working sheet is as follows.

- display stock data in different form (bar, polygon and spread sheet)
- provide different input methods through mouse and keyboard.

3) Identify inner objects within each interactive object.

Based on the interaction style of an interactive object, the interactive object can be decomposed into the following inner objects.

- one interaction object
- one linkage object
- one presentation object
- three display objects (for bar chart, polygon chart and spread sheet respectively)
- two lexical objects (one for bar chart and polygon chart, the other for spread sheet)

Hence, there are total eight inner objects in each interactive object and the configuration of the activated inner objects depends on the current state of the interactive object.

4) Specify the cluster controller for each interactive object.

There are three clusters of inner objects according to different states of the interactive object.

Bar chart state	Polygon char state	Spread sheet state
- interaction object	- interaction object	- interaction object
- linkage object	- linkage object	- linkage object
- presentation object	- presentation object	- presentation object
- bar chart display object	- polygon char display object	- spread sheet display object
- bar_polygon lexical object.	- bar_polygon lexical object	- spread sheet lexical object

There can be three possible clusters of activated inner objects but only one at a time. That is there can be three active message paths for the inner objects in the interactive object according to the internal current state of the interactive object. The configuration of this three clusters of inner objects can be shown in Figure 4.2, where L_obj1 is spread sheet lexical object, L_obj2 is bar_polygon lexical, D_obj1 is spread sheet display object, D_obj2 is polygon chart display object, D_obj3 is bar chart display object.

5) Specify each inner objects within an interactive object

Interaction object -

As interaction object falls on the semantic layer, it has partial semantic knowledge about an application. It is its duty to communicate

with the application through a linkage object. It handles the incoming message for updating, retrieving and predicting stock data. As retrieve and predict functions can be "undo", the interaction object need to push an undo object for reversing this operations into a undo stack before it performs these two operations.

A minimum safety value which is obtained form the application for giving semantic feedback to user is stored in this interaction object too.

Linkage object -

It translates the three command messages (updating, retrieving and predicting) from the interaction object and then sends them to the application side. In other way around, it also receives data from the application and decomposes them into stock data and minimum safety value and then sends them to the activated interaction object. All DDE communication processes are done in the Linkage Object and are hidden from other objects.

Besides for message translation, such as filtering out unnecessary information and collecting necessary information for both sides, an Linkage Object also helps to provide semantic feedback which is usually ignored by application.

Application only sends stock data to UI on request. However due to the semantic feedback, the UI also need the minimum safety value; hence, the linkage component at the application side also needs to pack the minimum safety value together with the stock data and sends them to UI.

Presentation object -

It stores all necessary syntactic and lexical information for presenting the information in an interactive object to a user in different views. It raises flags to inform display objects to give syntactic feedbacks and also maintains auto-scaling of the chart vertical axis. It also communicates with the interactive object cluster controller so as to update the activated inner objects. It maintains the consistency of enable and disable commands on a pull down menu.

Bar_Poly lexical object -

It gives lexical feedbacks to users. It changes the cursor shape when it receives a mouse button pressed and mouse moved signal when the stock data are displayed in a bar or polygon chart. Any other incoming messages that cannot be handled by this object is passed to its upper level object, presentation object.

Spread sheet lexical object -

Similarly to the above lexical object, it gives lexical feedback to users. It shows current editing cell when it receives a mouse button pressed, mouse moved and key pressed signal when the stock data are displayed in a spread sheet form.

Bar, poly and spread sheet display object -

They contain all necessary information and methods to display stock data in bar chart, polygon chart or spread sheet form. These objects only display stock data in different form and are not supposed to handle any incoming message from user.

6) Specify each rule in an event handler.

The pre and post conditions for each rule action are first specified according to the dialogue control in each inner object. Then the pre conditions are used to implemented the flags and events in the condition part of a rule; while the post conditions is used to implement the actual algorithm of the action part of a rule.

7) Specify interaction knowledge base.

The interaction knowledge base has three internal states for monitoring which interactive object is activated or deactivated and hence can provide message routing address for the kernel. It also contains a clipboard for stock data and used as an exchange buffer for the three interactive objects during "copy" and "paste" operations. It also raises flags to inform each interactive object to maintain output consistency after each dialogue switching.

Actually the inner objects form a U-tube ladder as shown in Figure 3.9. Any incoming message that cannot be handled by an object itself will be passed to its upper level objects. The hierarchy starts from lexical object as the lower and interaction object as the upper. As lexical object only handles low level operations, it should receive the user's inputs first. Interaction objects have partial semantic knowledge of the application and are supposed to perform high level operations. The inner objects can be viewed in such a way that the lower level inner objects (such as lexical object and presentation object) filter out all unnecessary messages for their upper level inner object (such as interaction

objects and linkage objects) because all the unrelated messages are already processed by the lower level inner objects. Therefore, the upper level objects can ignore the messages that are handled by the lower level inner objects. In the other way around, as the lower level inner objects will pass any unrecognized incoming messages to their upper level inner objects, they also ignore the messages that will be handled by its upper level inner objects. For example, the lexical object never needs to handle the messages for updating, retrieving and predicting stock data and interaction object never knows about the mouse button pressed or mouse moved message. Besides releasing the loading of message handling for each inner objects, such mutual ignorant mechanism also makes the inner objects become more interchangeable and hence it encourages software IC construction.

5.2.2 UI features supported by SSMDSS

The UI of the SSMDSS can support the following features:

1) Separation of UI and application

Physically, UI and application are totally separated as they are developed into two separate window programs and can only communicate with each other through the DDE mechanism. Actually, an application can serve several different UIs at the same time. Logically, the application ignores the interaction style between a users and the system. The linkage object between them compromises the differences between them and try to satisfy the needs of both sides by translating messages between them.

2) Multiple feedbacks on lexical, syntactic and semantic level.

Lexical feedbacks - give feedbacks for mouse button pressed, key pressed

and mouse moved signals by changing the cursor shape and position.

lexical and syntactic feedbacks -

Continuously show the stock value at the mouse cursor position as the mouse is drafting around.

semantic feedback - A warning is given to a user when the stock data below the minimum safety value.

3) Multi-thread dialogue among the three working sheet interactive objects.

4) Undo functions for different operations (retrieve, predict, copy, paste, and stock data entry using mouse or keyboard) through a single generic undo stack.

5) Provide different paths for a user to interact. (e.g. input stock value either by keyboard or mouse)

6) Consistency maintenance.

UI can maintain the consistency of its output when the system changes its state. For examples, the UI maintains the consistency of enabling and disabling command items on a pull down menu after each change of UI state. It also maintains the auto-scaling of the chart vertical axis according to the current stock values.

Chapter 6

Results

In this chapter, some observations of the SSMDSS implementation are presented. Some technical problems about the implementation in MicroSoft (MS) Window environment are stated. The accomplishments of the Object-Oriented UI Model indicated in this implementation are identified.

6.1 Facts discovered

6.1.1 Asynchronous and synchronous communication between objects

In the UI framework, all message passing through the kernel is asynchronous. A message sending object does not wait for the response of a message receiving object and continues its operation after sending the message to the kernel event queue. However, in some situations, the message sending object has to wait for the response from the message receiving object before it can continue its operation. This kind of communication is synchronous. In the SSMDSS, as automatic message routing for the synchronous communication cannot be done through kernel, in order to support automatic message routing for synchronous communication, the message sending object has to access the object pointer in the interaction knowledge base. Therefore, unlike the situation described in Figures 3.8 and 3.9, besides interaction objects, other objects may also interact directly with the interaction knowledge base for synchronous communication between objects.

The above phenomenon occurs because of the limitations of MS window

environment. As MS Window cannot provide us a real concurrent environment, synchronous communication can only be implemented as routine call rather than rendezvous as ADA does. Through the dynamic binding feature provided by the C++ language, synchronous communication, which is also automatic message routed, can be done by accessing object pointers in the interaction knowledge base directly. However, each object has to deal with two objects for message communication. One is the kernel object for asynchronous communication and the other is the interaction knowledge base for synchronous communication. If a system can be implemented in a real concurrent environment, both synchronous and asynchronous communication can be monitored by the kernel only. In such case, each object can be more interchangeable in deferent systems.

The pseudo concurrent environment in MS Window also requires each object in a UI to have the responsibility to release execution control back to the kernel of the UI framework. Therefore, the event handler in each object cannot contain any infinite loop as most real concurrent objects do.

6.1.2 Flexibility of C++ language

As C++ is a multi-paradigm language, a mixture of object oriented and non-object oriented programming, a non-object oriented paradigm program can be upgraded to an object oriented paradigm program in an easier and comfortable way. For example in the development of SSMDSS, we need to use MS Window Software Development Toolkit to call some window routines in the toolkit library. However, the routines in the toolkit are in non-object oriented paradigm. For instance, we cannot send a message to an object in the

development toolkit and require the object to do something for us. No overloading or polymorphism can be applied to the window procedures in the development toolkit. In order to use them, we have to call them in traditional C programming paradigm. C++ language can provide us such a flexibility to deal with this problem and at same time allow programmers enjoy the benefits of programming provided by Object-Oriented Programming. However, the mixture of programming may introduce many ad hoc programming which makes the software implementation cannot fully discharge the original object oriented design.

6.2 Technical Problems Encountered

6.2.1 Problem from Implementation Platform

In the MS Window environment, the "constructor" of a global object cannot be executed (a bug of Zortech C++ in MS Window environment). In order to tackle this problem, all global objects, such as the kernel or the interaction knowledge base, are declared as object pointers rather than objects themselves. Then somewhere in the main execution code, these object pointers are explicitly allocated. As all global objects are manipulated through pointers together with the object pointers for automatic message routing through dynamic binding, the code of UI may become clumsy and difficult to trace. In order to make the source code easier to read and trace, macro is used to replace the clumsy pointer manipulation syntax.

6.2.2 Problem due to Object Decomposition in an Interactive Object in SSMDSS

Functionally, the SSMDSS implementation, the Presentation, Lexical and Display Objects can be actually merged into a single object. However, in order to demonstrate the feature of the Object-Oriented UI Model, we have decomposed it into several objects as described in chapter 5. However, such object decomposition may cause the following problem.

An object decomposition may cause the difficulty of determining how much information should be embedded in each object and which objects should fall on which linguistic layer. For example in the above case, Presentation Object is supposed to contain all necessary information to present an interactive object. However, Display Objects and Lexical Objects may also need this information to display output and give feedback to a user. If this information is declared as "public" and we allow other objects such as Display Objects and Lexical Objects to access it directly, information hiding will be lost and hence this object decomposition will cause an obstacle for software IC constructor. However, if the information is accessed through message passing, it will increase the loading of event handler in each object. Finally, if we duplicate the information in each object, it may also cause information inconsistent and updating problem. Due to above difficulties, in the implementation of SSMDSS, in order not to increase the loading of event handler in each object, the information in Presentation Object is declared as "private" but can be accessed by the Lexical Objects and Display Objects through the "friend" features in C++. By doing so, we sacrificed a certain degree of software IC construction.

Hence, an object decomposition may introduce a trade off between

information hiding which can encourage software IC construction and efficiency of co-operation among objects. In order to resolve the above conflict, balance between information hiding and co-operation among objects should be carefully maintained.

6.3 Objectives accomplished by the Object-Oriented UI Model indicated by the SSMDSS

Separating UI from application

SSMDSS is a good example to demonstrate the dialogue independent concept because, through the Linkage component, application totally ignores the computer-user interaction such as different presentations of data, different input methods through mouse or keyboard, feedbacks from different linguistic levels and undo functions for different reversible operations.

Multiple Continuous feedbacks

Multiple feedbacks are given to a user by objects at different level. Lexical feedback that changes the shape of the cursor when a mouse button is pressed is given by a Lexical Object; lexical and syntactic feedbacks that continuously showing the stock value at the current mouse cursor position is given by a Presentation Object.

Multi-thread Dialogue

Dialogue switching among the three working sheets in SSMDSS through a simple switch box mechanism described in section 4.1.4 demonstrates the feature of multi-thread dialogue.

Automatic Message Routing

With the help of the kernel and the interaction knowledge base, each object in the SSMDSS ignores the destinations of its outgoing messages.

UNDO / Recovery

The single generic UNDO stack in each working sheet interactive object provides undo functions for retrieve, predict, copy, paste, and stock data entry via mouse or keyboard operations in the SSMDSS. Note, this undo functions are at different abstract levels but they are all processed through a single UNDO stack.

Maintaining Consistency

Consistency of the command menu in the SSMDSS is maintained by the ϵ-rules described in section 3.4.2.

Software IC Construction (not fully support)

Due to the nature of the SSMDSS and the problem described in section 6.2.2, software IC construction in the SSMDSS is not fully supported. However, automatic message routing provides certain amount of software IC construction in the SSMDSS. At least, adding or removing interactive objects does not interfere with other objects.

Systematic Method to Specify and Develop UIs (not shown)

As our Object-Oriented UI model was continuously revised during the implementation of SSMDSS, there is no evidence to show that this objective is accomplished in the implementation of SSMDSS. However, the specification approach and steps described in section 3.4 and 3.5 may contribute this objective in some degree.

Chapter 7

Conclusion

7.1 Thesis Summary

Features in modern UIs, such as direct manipulation and multi-thread dialogue, introduce new problems to UI development. The objective of this thesis is to provide solutions to these problems so that the software development and maintenance cost of modern UIs can be reduced. An Object-Oriented UI Model and a UI Framework whose design is based on this concept are proposed in this thesis. They provide a new development approach to modern UIs such that the following development qualities can be accomplished:

1) Separating UI from application
2) Multiple Continuous feedbacks
3) Multi-thread Dialogue
4) Automatic Message Routing
5) UNDO / Recovery
6) Maintaining Consistency
7) Software IC Construction
8) Systematic Method to Specify and Develop UIs

A Simple Stock Market Decision Support System (SSMDSS) is implemented based on the Object-Oriented UI Model in MS window environment. The development of the SSMDSS was studied. All of the above development objectives, except points 7 and 8, are achieved in the development of SSMDSS.

7.2 Merits and Demerits of the Object-Oriented UI Model

Merits:

The UI Framework provides a basic blue print for UI development so as to reduce the UI development time and hence encourages UI rapid prototyping. The Linkage component realizes dialogue independence in an application. Dialogue switching between interactive objects can be easily achieved by a simple switch box mechanism. Multiple continuous feedbacks can be given automatically by inner objects in an interactive object. The automatic message routing supported by the kernel and the interaction knowledge base encourages information hiding and hence facilities easy modification of UI. The Undo / Recovery mechanism provided by the model releases the inconvenience of the interactive objects for handling undo operations at different abstract levels through a single UNDO stack. The model also proposes a new approach to specify objects in a UI so that the dialogue control and consistency maintenance can be easily specified and implemented based on the UI framework. Demerit:

In order to reduce the loading of the Linkage component and to increase the efficiency of the co-operation between a UI and its application, both the UI and its application should be developed using an object-oriented paradigm. Such limitation restricts the flexibility of the application design. However, as object-oriented design becomes more and more common and prevalent in software development, it is believed that the above limitation can be gradually eliminated in the future.

7.3 Cost of Object-Oriented UI Model

Although the Object-Oriented UI Model can shorten the UI development time, it degrades the UI run time performance. That is, a UI based on the Object-Oriented UI

Model certainly runs more slowly and occupies more memory than the one based on the traditional software development providing that they both have the same UI capability. The degradation is due to the overhead of automatic message routing and message translation between UI and application through the Linkage component. In the view point of machine execution, automatic message routing is much more expensive than a direct routine call. The degradation becomes further severe when the application side is not developed by using an object oriented paradigm as the Linkage component have to need more computation power to perform the message translation.

Despite the above degradation, the Object-Oriented UI Model should be still justified. Although the run time performance is degraded, this degradation is insignificant to human response. User can not tell if a UI slows down for several milliseconds. In addition, as the price of graphics hardware is kept going down and on the contract the software cost is kept going up, it is justified to shorten the UI development time on the expense of hardware cost.

7.4 Future Work

In the SSMDSS implementation, the interaction knowledge base is mainly used as a global controller for message routing between objects and dialogue switching between interactive objects in UI. In the future, we can add more rules and constraints in the interaction knowledge base so that it can increase the co-operations, linkage and consistencies between interactive objects such as the copy and paste commands in the SSMDSS.

One of the future work of the Object-Oriented UI Model is to include the Vienna Development Method (VDM) [34,47,48] so as to specify a UI in a formal, systematic and

mathematic approach. Based on this formal UI specification, a UI specification interpreter can be developed so that UI specification can be executed under the interpreter before the UI is actually implemented. The execution of UI specification is also a key success of UI rapid prototyping.

Appendix

A1 An Algorithm for Converting Transition Network Diagram to Event Response Language

Event Response Language (ERL) is suitable for modeling dialogue control in World Model style user interfaces because it can describe asynchronous and multi-thread dialogue. However, as ERL is multi-thread in nature and hence it is difficult for us to capture the user's input sequences from the ERL. On the other hand, Transition Network Diagram (TND) can clearly describe the user's input sequences but TND is not suitable for our Object-Oriented UI model. Fortunately, there is an algorithm [29] to convert Transition Network Diagram to ERL. Therefore, we can specify UI dialogue control in TND first in the early UI specification and then we convert the TND into ERL which is easier to be implemented under the Object-Oriented UI model environment.

M. Green [29,30] has proposed an algorithm to convert a TND into an ERL and has shown that the description power of the ERL is not lesser than the original TND. In this thesis, some notation and steps of the algorithm are modified and simplified so that the algorithm can be applied to our Object-Oriented UI model efficiently. For instance, subdiagrams in a transition network corresponds to an event handler in other inner objects. After the conversion, event handlers do not need "active" flags to indicate which event handler is currently active because this problem has already been resolved by the switch box mechanism and automatic message routing in our Object-Oriented UI model (see section 3.4.5 and 4.1.4). Below we only list the modified algorithm used in this

thesis. Readers who are interested in the original algorithm can refer [29]. Although some notations and steps of the original algorithm are modified and simplified, the basic methods and concepts are unchanged.

1) The first step in the conversion algorithm is to calculate all LEADING relations of each subdiagrams in the TND. The relation LEADING can be defined as:

$$\text{LEADING}(d) = \{a \mid a \in \Sigma \text{ and } aS \in L(d)\}.$$

Where S = input string

d = subdiagram in the TND

L(d) = set of strings in Σ that are recognized by d.

That is, every string in L(d) labels a path from the initial state to one of the final states of d.

This LEADING(d) is used to construct outgoing messages in an event handler template. (see section 3.4)

2) The second step is to construct all incoming messages in the event handler by collecting all input tokens that are labeled on each arc in the TND. If the arc is labeled by a subdiagram name, then all the tokens in the LEADING set for that subdiagram are also collected.

The incoming messages = input tokens on each arc label

+

LEADING for each subdiagram

The incoming messages are used to construct the CONDITION part of regular rules in the event handler template.

3) The third step is to construct all regular rules for each incoming message obtained in step 2. The CONDITION part of each regular rule consists of an incoming message and a triggering flag which corresponds a state in the TND. The ACTION part of a regular rule consists of an action that will be executed when a arc in the TND is traversed or/and a state updating process, if necessary, for updating the triggering flag of CONDITION part in other regular rule.

There is set of regular rules for each state in the TND and within this set of regular rules, there is a regular rule for each outgoing arc label of that state. Therefore, after the conversion, the number of regular rule in an event handler will not be lesser than the number of arc in the corresponding TND.

	CONDITION	--------->	ACTION	
ERL	Incoming message	Triggering flag	Actions	State updating processes
TND	Input token on arc label	Current state	Actions to be executed when the arc is traversed	next state after the arc is traversed

Table A1.1 Comparison of components for each notation.

4) The fourth step is to construct all event handlers in each inner object according to the above steps.

A2 An Object-Oriented Software Development

A2.1 Traditional Non Object-Oriented Software Development

In traditional non Object-Oriented software development, the software life cycle, in general, consists of four phases: Analysis, Design, Implementation and Testing. These phases are considered as some linear series of development processes. Each of these processes must be completed before the next is commenced.

The Analysis phase initiates the software development. It defines user requirements and identifies problem scope. It also includes feasibility of the project development. This stage figures out "WHAT" system should be built.

The Design phase covers system design, logical design and detailed design for implementation. It tells system developers "HOW" to build the system.

The Implementation stage actually implements the system according to the design specification obtained from the previous stages.

The Testing stage covers units testing, system testing, verification and validation of the system. This stage makes sure the final product completely fulfil our client requirements.

In traditional software development, the development processes emphasize some identifiable activities and their functional decompositions. The system analysis and design concentrate on "WHAT" does the system do and WHAT is its function. Functional decomposition is obviously a top-down analysis and design methodology. Through function decomposition, the translation of the problem space to solution space is based on an interdependent set of functions or procedures.

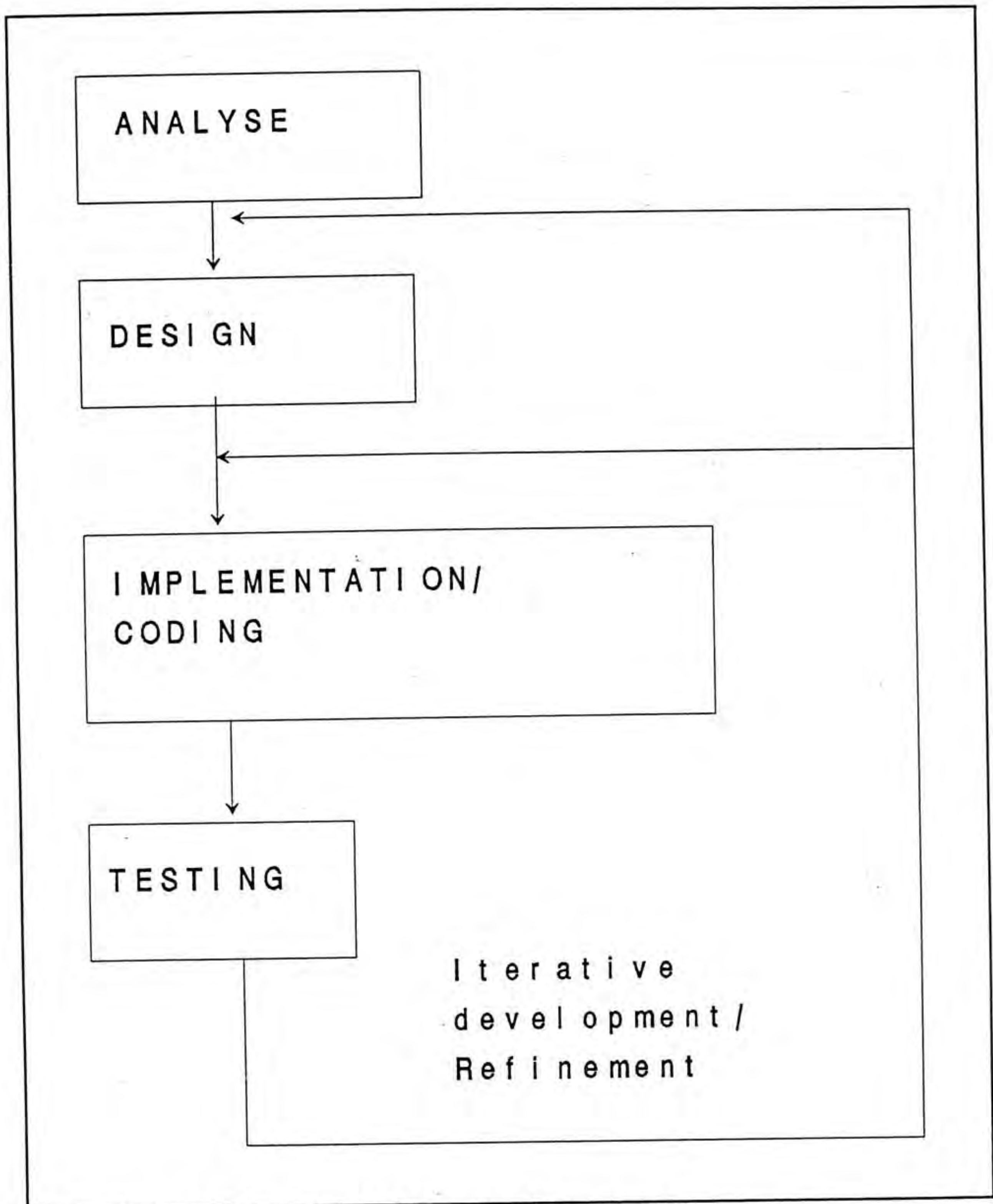


Figure A2.1 The traditional life cycle

The drawback of this development approach is the high cost of system refinement/maintenance. If some evolutionary changes occur at low level function design phase may cause great changes at the top level system design phase or even at system analysis phase. As a result, a small change at low level development stage may cause the development life cycle start over again from the

very beginning.

A2.2 An Object-Oriented Software Development

Unlike traditional software development, which stresses on functions or procedures, Object-Oriented software development emphasizes objects --- entities that encapsulate both data and procedural features together. Systems are viewed as a collection of objects rather than functions. The system decomposition is also done by object decomposition. The relation between objects can be specified by an E-R model. The control of a system is monitored by a message-object model in which messages are passed between objects and invoke procedures than embedded in objects. Objects communicate with each other through messages and play the roles of client or server rather than caller or callee in ordinary routine call. A server object responses the request of a client object according to its internal procedures. Services that the server object can provide are visible to the client object but HOW the server object responses the client object request is hidden from the client object. This kind of information hiding can allow us to defer detail design, such as procedure algorithm implementation and specification of data structure, during systems design. On the other hand, changes of detail design at low level do not interfere with the systems design at high level. Therefore, object oriented software development facilities system refinement in a software development life cycle.

Booch [6] identifies five major stages for Object-Oriented software development:

1) Identify objects and attributes

2) Identify operations affecting objects

3) Establish visibility

4) Establish interface

5) Implement each object

Our Object-Oriented UI model presented in this thesis also applies the above software development methodology. Step 1 corresponds to the identification of interactive objects and its inner objects in a UI. Step 2 corresponds to the identification of rules of an event handler in an object. Steps 3 and 4 correspond to the identification of incoming messages for each event handler in an object. Step 5 corresponds the implementation of rules in each event handler. The above Object-Oriented software development steps have been demonstrated in section 3.5 in this thesis.

A3 Vienna Development Method (VDM)

A3.1 An Overview of VDM

VDM has been applied to traditional software development for a long time. It provides both a specification notation and proof obligations which enable a designer to establish the correctness of design steps. It can give a formal, systematic and mathematic approach to specify and to develop software system so that the implementation of the system can fully discharge the original requirement specification. In brief, under the VDM, a system is developed according to the following steps:

1) Specify the system formally.

The system is specified by a set of operations applying to a set of (or a class of) valid states. The operations are further specified by pre-condition predicates and post-condition predicates; while the states is defined by data type invariant. By using this form of specification, the initial specification only captures abstract concepts of the system and avoid implementation details.

2) Implementability proof obligation.

Prove that individual operation can be implemented from the pre and post condition of the operation. (Implementability proof obligation)

Do

3) Realization of specification.

The specification is refined by including implementation detail. This step can be done by either data reification or operation decomposition.

4) Prove that the realization satisfies the previous specification.

Until the realization is as concrete as program.

Under the VDM, there are four proof obligations for software development:

i) Implementability proof obligation

$$\forall \overleftarrow{\sigma} \in \Sigma \cdot pre\text{-}OP(\overleftarrow{\sigma}) \Rightarrow \exists \sigma \in \cdot post\text{-}OP(\overleftarrow{\sigma}, \sigma)$$

Where pre-OP	= Pre-condition of operator OP
post-OP	= Post-condition of operator OP.
$\overleftarrow{\sigma}$	= initial state before the operation OP.
σ	= state after the operation OP.

This proof obligation is to check if the operator OP can be implemented. If the operator can be implemented, then there must exist a final state such that the post condition of the operation can be satisfied.

ii) Adequacy proof obligation

$$\forall a \in A \cdot \exists r \in R \cdot ret(r) = a$$

Where A	=	Abstraction set
R	=	Representation set
ret =		Retrieve function to transform representations of type R to representations of type A

This proof obligation asserts that every possible state value in the abstract model has at least one representation in the reified model.

iii) Operation modelling proof obligation - domain rule

$$\forall r \in R \cdot pre\text{-}OPA(ret(r)) \Rightarrow pre\text{-}OPR(r)$$

Where OPA = abstract operation

OPR = reified operation

This proof obligation asserts that the pre-condition of the abstract operation should satisfy the pre-condition of the reified operation.

iv) Operation modelling proof obligation - result rule

$$\forall \overleftarrow{r}, r \in R \cdot$$

$$pre\text{-}OPA(ret(\overleftarrow{r})) \wedge post\text{-}OPR(\overleftarrow{r}, r) \Rightarrow post\text{-}OPA(ret(\overleftarrow{r}), ret(r))$$

This proof obligation asserts that the initial state satisfies the pre-condition of the abstract operation and that the state pair satisfy the post-condition of the reified operation then the two states will produce a state pair that will satisfy the post-condition of the abstract operation.

For more literatures about VDM, readers can refer [34,47,48].

A3.2 Apply VDM to Object-Oriented UI model

As we can see, the software development under VDM starts with operators and states. Data reification and operation decomposition can be refined separately. This development environment is not matched with the object-oriented paradigm which encapsulates data and methods into a single entity called object. In order to include VDM in object-oriented design environment, the above

development steps may have to be modified into the following steps:

1) Specify the system formally.

The system is specified by a set of objects. The objects are further specified by a set of attributes (or states) and methods (operations) on them. Pre and post condition can still be used to specify each method within objects. The object relation and hierarchy should be specified in this step too.

2) Implementability proof obligation.

Now, the proof concerns not only about the pre and post conditions of methods within the object itself but also concerns about the pre-condition of the object instantiation and the ones in its superclass too. The situation is further complicated if the object is allowed multi-inherent.

Do

3) Realization of specification.

The refinement should be done by object decomposition and specialization. Data reification or operation decomposition which can be applied to the attribute and methods within an object should be done for each object decomposition or specialization.

4) Prove that the realization satisfies the previous specification.

Until the realization is as concrete as program.

We also need to add some simple syntax into the VDM so that it can represent some basic object oriented features such as inheritance and polymorphism.

A4 Glossaries and Terms

Abstract Base Classes

- They define all the basic components in a framework. They also define the standard interfaces for their derived class objects. Polymorphism among objects is done through this standard interfaces defined in the abstract base classes. See section 4.1.1.

Aggregation (composition mechanism)

- One of the abstraction mechanism of Object-Oriented paradigm. Aggregation refers to how certain model constructs may be viewed as collections or aggregates of the other model constructs. Relationships between low-level types can be considered a higher level type.

Attributes

- Each object can have certain number of attributes (states of an object) and each attributes can be an object of other types.

Automatic Message Routing

- It is a mechanism that the message routing is automatically done by the UI kernel and is transparent to an message sending object. See section 3.4.5.

Constructor

- A feature of C++ programming. Each object can have its own constructor in which all the statements in the constructor will be executed when the object comes into existence. Constructor can accept input parameters during the object instantiation. Constructor can be overloaded.

Conversational World Style User Interface

- This kind of user interface treats human computer interaction as human conversation in which each participant speaks in turn. This style of interaction is usually adopted by most conventional text-based interfaces. See section 2.1.

Class Hierarchy

- Each class has one or more subclass/superclass. Superclass defines more general behavior of the objects while the subclass defines more specific behavior of the objects.

Classes

- A class defines the behavior of similar object. Every object belong to a class has the same method and data structure. Every object is an instance of a class (or a member of a class).

Classification

- One of the abstraction mechanism of Object-Oriented paradigm. Most objects have a similar structure and share a common set of properties. Classification allows one to ignore the details of particular objects by using a

construct which represents a set of objects with a similar structure.

Dialogue Independence

- A requirement for separating user interface from application. It is a design approach in which application should not depend on any human-computer dialogue style in its user interface. In the other way around, the interaction style in a user interface should not depend on the computation algorithm in the application. See section 3.1 and 3.2.

Dynamic Binding

- A feature of C++ programming. Through dynamic binding we can defer the code binding of a procedure call until at the moment of the call at run time. Dynamic binding is a realization of polymorphism in Object-Oriented paradigm.

Encapsulation (binding of data and methods)

- Data and methods (procedures) are encapsulated in an abstract unit call OBJECT. You cannot apply any method to the data of an object, you can only require the object to manipulate its data through its own method.

Event

- An event signals a change (something has happened) in human computer interaction. A user can monitor the interaction through some input devices. Therefore this input devices can be considered as sources of events. New events generated from this sources are first put into an event queue and then they are delivered to their proper destinations. In this thesis, the delivery of events is done by the kernel in the UI framework through the automatic message routing mechanism.

Event Handler

- A basic method in an object. It handles all incoming messages. Event handler is constructed by rules and is modeled by Event Response Language.

Event Response Language

- A notation for describing human-computer dialogue. The main elements of Event Response Language are incoming events, outgoing events, and flags. These elements can be used to build rules in an event handler.

Framework

- It provides a basic foundation for a software system development. See section 4.1.

Generalization/Specialization

- One of the abstraction mechanism of Object-Oriented paradigm. Generalization refers to the formation of a single class by combining two or more distinct classes. Differences among similar objects in the classes are ignored to form a higher order class in which the similarities can be emphasized. Specialization is the inverse of generalization, which is used to generate new classes.

Generic UNDO Stack

- A stack which contains UNDO objects for UNDO operation or system recovery. This generic UNDO stack can contains any type of UNDO objects regardless of what their UNDO operations are. See section 3.3.7.

Inheritance

- Every object can inherit all properties of its superclass (including the superclass method and data)

Interaction Knowledge Base

- A reservoir of all global states in a user interface. It monitors which interactive object is activated or de-activated. It also helps the kernel to conduct automatic message routing. See section 4.1.3.

Interactive Object

- One of the basic components in our Object-Oriented UI model. It is an object that a user can conduct a meaningful step-by-step dialogue and user can apply direct manipulation on it. Dialogue switching in multi-thread dialogue is done between these interactive objects. An interaction object may consist of several inner objects, such lexical objects, display objects, ...etc.

Linkage

- It is a intermediary between an application and its user interface. It translates messages for both sides. With the help of Linkage component, application and user interface can be developed separably. Linkage model is a realization of separating user interface from application. See section 3.2.

Model World Style User Interface

- This style of interaction tries to represent real world objects visually so that a user can manipulate this object directly through some input devices such as mouses or light pens. See section 2.1.

Multi-thread Dialogue

- One of the features of model world style interaction. A user can interact with several objects at a same time. The user is free to switch from one dialogue to another at any point in the interaction. See section 2.1.

Multiple Continuous Feedbacks

- It is a kind of human-computer interaction in which feedback are given to a user from different levels (lexical, syntactic and semantic) continuously. The multiple feedbacks are continuously given to the user even the user has not finished his command. See section 3.1 and 3.3.6.

Object

- Object is the basic component in Object-Oriented paradigm. Object ia an entity that encapsulates both data and procedural features together. Object responds incoming message according to its internal data and procedures.

Polymorphism

- Different objects can respond to the same message with their own unique behavior differently. As a method in a subclass can override the same name method in its superclass, a subclass object and a superclass object can respond the same message differently.

Software IC

- One of the objectives of Object-Oriented software development. In Object-Oriented paradigm, each object can be considered as an Integrate Circuit (IC) in hardware design in which an IC can be easily replaced by other IC without interfering with other components in a system. Software IC concept facilities iterative design and software maintenance.

Specialization

- See Generalization.

Specification (interface) and implementation of an object

- There are two parts in an object:

i) Specification part -- This is the visible part of the object. It tells a user what kind of methods and data that he can request or access. This part is public to all other object.

ii) Implementation part -- This part is only visible to its object itself. All the methods and data declared in this part are only accessed by the objects itself. This part is private to all other object.

Switch Box Mechanism

- A mechanism for dialogue switching in multi-thread dialogue. Through this mechanism, only a single parameter is need to be updated in order to switch a dialogue control between interactive objects.

Reference

[1] Allen, R.B. Cognitive Factors in Human Interaction with Computers. *Directions in Human Computer Interaction*, edited by A. Badre and B.Shneiderman. Ablex ,New Jerser, 1982, 1 - 26.

[2] Barth, P.S. An Obejct-Oriented Approach to Graphical Interfaces. *ACM Transactions on Graphics.* Vol. 5, No. 2, April 1986, 142 - 171.

[3] Bershad, B.N., and Levy, H.M. A Remote Computation Facility for a Heterogeneous Environment. *Computer*, (May 1988), 50-60.

[4] Bershad, B.N., Ching, D.T., Lazowska, E.D., Sanislo, J., and Schwartz, M. A Remote Procedure Call Facility for Interconnecting Heterogeneous Computer Systems. *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 8 (Aug. 1987), 880-894.

[5] Birrell, A.D., and Nelson, B.J. Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems*, Vol 2, No. 1, (FEB. 1984), 39-59.

[6] Booch, G. Object-Oriented Development. *IEEE Transactions of Software Engineering,* Vol.SE-12, No.2, February 1986, 211 - 221.

[7] Coad, P. and Yourdon, E. *Object-Oriented Analysis*. Prentice-Hall. 1990.

[8] Cockton, G. Interaction ergonomics, control and separation: open problems in user interface management. *Information and Software Technology,* Vol.29, No. 4, May 1987, 176 - 191.

[9] Cockton, G. A New Model for Separable Interactive Systems. *Human-Computer Interaction - INTERACT'87*, edited by H.J. Bullinger and B. Shackel, North-Holland, 1987, 1033 - 1038.

[10] Cockton, G. Where do we draw the line? - Derivation and Evaluation of User Interface Softwarre Separation Rules. *People and Computers: Designing for Usability*, edited by M.D. Harrison and A.F. Monk, Cambridge University Press, (1986), 417 - 431.

[11] Coutaz, J. The Construction of User Interfaces and the Object Paradigm. *ECOOP '87 European Conference on Object-Oriented Programming*, edited by Bezivin, (1987), 121-130.

[12] Coutaz, J. Abstractions for User Interface Design. *Computer*, September 1985, 21 - 34.

[13] Cox, B.J. *Object-oriented programming*. Addison-Wesley, 1987.

[14] Cox, B. and Hunt, B. Objects, Icons, and Sotfware-ICs. *BYTE.* August 1986, 99 - 108.

[15] Dillon, A. A PSYCHOLOGICAL VIEW OF "USER-FRIENDLINESS". *Human-Computer Interaction - INTERACT' 87*, edited by H.J. Bullinger and B.Shackel, North-Holland, 1987, 157 - 163.

[16] Durant, D., Carlson, G. and Yao, P. *Programmer's Guide to Windows.* SYBEX, 1987.

[17] Eckel, B. *Using C++.* McGraw-Hill. 1989.

[18] Enderle, G. Report on the Interface of the UIMS to the Application. *User Interface Managment System*, edited by G. E. Pfaff, Springer-Verlag, (1983), 21-29.

[19] Farooq, M.U. and Dominick, W.D. A survey of formal tools and models for developing user interfaces. *Journal of Man-Machine Studies.* 29, 1988, 479 - 496.

[20] Fischer, G. and Lemke, A.C. Constrained Design Processes: Steps Towards Convivial Computing. *Cognitive Science and its Application for Human-Computer interaction*, edited by R. Guindon. Lawrence Erlbaum Associates, New Jersey, 1988, 1 - 58.

[21] Foley, J., Kim, W.C., Kovacevic, S. and Murray,K. Defining Interfaces at a High Level of Abstraction. *IEEE Software.* January 1989, 25 - 32.

[22] Foley, J. Transformations on a Formal Specification of User-Computer Interfaces. *Computer Graphics*, Vol.21, No. 2, April 1987, 109 - 115.

[23] Foley, Van Dam, Feiner, Hughes. *Computer Graphics Principles and Practice.* Addison Wesley, 1990.

[24] Gibbs, S. An object-orinted office data model. *Technical Report CSRG-154*, University of Toronto (1985).

[25] Gibbons, P.B. A stub Generator for Multilanguage RPC in Heterogeneous Enivronments. *IEEE Transactions on Software Engineering*, Vol, SE-13, No. 1, (Jan 1987), 77-87.

[26] Gimnich, R and Ebert, Jurgen. Constructive Formal Specifications for Rapid Prototyping. *Human-computer Interaction - INTERACT' 87*, edited by H.J. Bullinger and B. Shackel, North-Holland, 1987, 1047 - 1052.

[27] Goodwin, M. *User Interfaces in C++ and Object-Oriented Programming.* MIS Press, 1989.

[28] Gray, P.D., Kilgour, A.C. and Wood, C. A. Dynamic reconfiguawbility for fast

prototyping of user interfaces. *Software Engineering Journal.* November 1988, 257 - 262.

[29] Green, M. A Survey of Three Dialouge Models. *ACM Transactions on Graphics.* Vol. 5, No.3, July 1986, 244 - 275.

[30] Green, M. The University of Alberta User Interface Management System. *SIGGRAPH'85.* Vol.19, No.3, 1985, 205 - 213.

[31] Grossman, M. and Ege, R.K. Logical Composition of Object-Oriented Interfaces. *OOPSLA'87 Proceedings.* October 4 - 8, 1987, 295 - 306.

[32] Hartson, H.R. and Hix, D. Human-Computer Interface Development: Concepts and Systems for Its Management. *ACM Computing Surveys.* Vol.21, No.1, March 1989, 5-92.

[33] Hartson, R. User-Interface Management Control and Communication. *IEEE Software*. January 1989, 62 - 70.

[34] Hekmatpour, S. and Ince, D. *Software Prototyping, Formal Methods and VDM.* Addison-Wesley. 1988.

[35] Henderson-Sellers, B. and Edwards, J.M. The Object-oriented Systems Life Cycle. *Communcication of the ACM*. Vol. 33, No.9, September 1990, 143 -159.

[36] Hill, R.D. Some Important Features and Issues in User Interface management Systems. *Computer Graphics*. Vol. 21, No.2, April 1987, 116 - 119.

[37] Hill, R.D. Supporting Concurrency, Communication, and Synchronization in Human-Computer Interaction - The Sassafras UIMS. *ACM Transaction on Graphics*. Vol. 5, No.3, July 1986, 179 - 210.

[38] Hu, D. *Object-Oriented Environment in C++*. MIS Press, 1990.

[39] Huang, K.T. Visual Interface Design Systems. *Principles of Visual Programming System*, edited by S.K. Chang, Prentice-Hall, 1990, 60 - 143.

[40] Hudson, S.E. and King, R. Semantic Feedback in the Higgens UIMS. *IEEE Transactions on Software Engineering.* Vol. 14, No. 8, August 1988, 1187 - 1206.

[41] Hudson, S.E. UIMS Support for Direct Manipulation Interfaces. *Computer Graphics*. Vol.21, No.2, April 1987, 120 - 124.

[42] Hudson, S.E. and King, R. A Generator of Direct Manipulation Office System. *ACM Transactions on Office Information Systems*. Vol.4, No.2, April 1986, 132 - 163.

[43] Hurley, W.D. and Sibert, J.L. Modeling User Interface-Application Interactions. *IEEE Software*, January 1989, 71 - 77.

[44] Jacob, R.J.K. A Specification Language for Direct-Manipulation User Interfaces. *ACM Transactions on Graphics*. Vol. %, No. 4, October 1986, 283 - 317.

[45] Jacob, R.J.K. An Executable Specification Technique for Describing Human-Computer Interaction. *In Advances in Human-Computer Interaction*, H.R. Hartson, Ed. Ablex, Norwood, N.J., 1985, 211 -242.

[46] Jacob R.J.K. Using Formal Specifications in the Design of a Human-Computer Interface. *Communications of the ACM*. Vol.26, No.4, April 1983, 259 - 264.

[47] Jones, C.B. and Shaw R.C. *Case Studies in Systematic Software Development*. Prentice Hall, 1990.

[48] Jones, C.B. *Systematic Software Development using VDM (2nd edition)*. Prentice Hall, 1990.

[49] Jones, W.P. "As We May Think" ?: Psychological Considerations in the Design of a Personal Filing System. *Cognitive Science and its Application for Human-Computer interaction*, edited by R. Guindon. Lawrence Erlbaum Associates, New Jersey, 1988, 235 - 288.

[50] Jordan, D. Implementation Benefits of C++ Language Mechanisms. *Communications of the ACM*. Vol.33, No.9, September 1990, 61 -64.

[51] Kamran, A. Issues Pertaining to the Design of User Interface Management System. *User Interfae Management System,* edited by G.E. Pfaff, springer-Verlag, 1983, 43 - 48.

[52] Koivunen, M., and Mantyla, M. HutWindows: An Improved Architecture for a User Interface Management System. *IEEE Computer Graphics & Applications.* (Jan 1988), 43 - 52

[53] Korson, T. and McGregor, J.D. Understanding Object-oriented : A Unifying Paradigm. *Communications of the ACM*. Vol.33, No. 9, September 1990, 40 - 60.

[54] Kurtz, D. B., Ho, D, and Wall. T.A. An Object-Oriented Methodology for System Analysis and Specification. *Hewlett-Packard Journal*, April 1989, 86 - 90.

[55] Lantz, K.A. Multi-process Structuring of User Interface Software. *Computer Graphics*. Vol.21, No.2, April 1987, 124 - 130.

[56] Lam, S. H. Rapid Prototyping of Interactive User Interface. *Term Paper of Department of Computer Science*, CUHK (1990).

[57] Lam, S. H. Separation of User Interface and Application in Object-Oriented Approach. *Term Paper of Department of Computer Science,* CUHK (1989).

[58] Lam, S.H. An Overview of Multimedia Message Handling in Office Automation. *Term paper of Department of Computer Science*, CUHK. (1989).

[59] Ledbetter, L. and Cox, B. Software-ICs. *BYTE*. June 1985, 307 - 316.

[60] Linton, M.A., Vlissides, J.M. and Calder, P.R. Composing User Interfaces with InterViews. *Computer.* February 1989, 8 - 22.

[61] Maguire, M.C. A Review of Human Factors Guidelines and Techniques for the Design of Graphical Human-Computer Interfaces. *Computer and Graphics,* Vol. 9, No. 3, 1985, 221 - 235.

[62] Martin, T.P. A communciation Model for Message Management System. *Technical Report CSRG-157,* University of Toronto, April 1984.

[63] McDonald, J.E. and Schvaneveldt, R.W. The Application of User Knowledge to Interface Design. *Cognitive Science and its Application for Human-Computer interaction*, edited by R. Guindon. Lawrence Erlbaum Associates, New Jersey, 1988, 289 - 338.

[64] Meyer, B. A. User-Interface Tools: Introduction and Surver. *IEEE Software.* January 1989, 15 - 23.

[65] Meyer, B. *Object-oriented Software Construction*. Prentice Hall, 1988.

[66] Myers, B. and Doner, C. *Graphics Programming Under Windows.* SYBEX, 1988.

[67] Meyer, B. Resuability: The Case for Object-Oriented Design. *IEEE Software,* March 1987, 50 - 60.

[68] Mullin, M. *Object Oriented Program Design*. Addison-Wesley, 1989.

[69] Notkin, D., Black, A.P., Lazowska, E.D., Levy, H.M., Sanislo, J., and Zahorjan, J. Interconncetion Heterogeneous Computer Systems. *Communication of the ACM*, Vol 31, No. 3, (March 1988), 258-273.

[70] Olsen, D.R., Dempsey, E.P. and Rogge, R. Input/Output Linkage in a User Interface Management System. *SIGGRAPH'85*, Vol.19, No.3, 1985, 191 - 197.

[71] Pascoe, G.A. Elements of Object-Oriented Programming. *BYTE.* August 1986, 139 - 144.

[72] Peatroy, D.B. and DATATECH Publications. *Mastering The Macintosh™ Toolbox.* McGraw-Hill, 1986.

[73] Polson, P.G. The Consequences of Consistent and Inconsistent User Interfaces. *Cognitive Science and its Application for Human-Computer interaction*, edited by R. Guindon. Lawrence Erlbaum Associates, New Jersey, 1988, 59 - 108.

[74] Rathke, M. Dialogue Issues for Interactive Recovery - an Object-Oriented Framework. *Human-Computer Interaction - INTERACT'87*, edited by H.J. Bullinger and B. Shackei, 1987, 745 - 750.

[75] Rhyne, J., Ehrich, R., Bennett, J., Hewett, T., Sibert, J. and Bleser,T. Tools and Methodology for User Interface Development. *Computer Graphics*. Vol.21, No.2, April 1987, 78 - 87.

[76] Rosson, M. B., Maass, S. and Kellogg, W.A. The Designer as User: Building Requirements for Design Tools from Desgin Practice. *Communication of the ACM.* Vol 31, No. 11, November 1988, 1288 - 1298.

[77] Sakkinen, M. On the darker side of C++. *ECOOP'88 Eurpopean Conference on Object-Oriented Programming*, edited by , North-Holland, 1988, 162 - 176.

[78] Schmucker,K.J. MACAPP: An Application Framework. *BYTE*. (August 1986), 72 - 75.

[79] Schmucker,K.J. *Object-Oriented Programming for the Macintosh™*. Hayden, 1986.

[80] Shneiderman, B. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley, 1987.

[81] Shneiderman, B. Direct Manipulation: A Step Beyond Programming Languages. *IEEE Computer*. August 1983, 57 - 69.

[82] Sibert, J., Belliardi, R and Kamran, A. Some Thoughts on the Interface Between User Interface Management System and application Software. *User Interface Managment System*, edited by G.E. Pfaff, Springer-Verlag, 1983, 183 - 189.

[83] Sibert,J.L., Hurley,W.D. and Bleser, T.W. An Object-Oriented User Interface Management System. *Computer Graphics*, Vol 20, No. 4, August 1986, 259-268.

[84] Simoes. L.P. and Marques, J. A. IMAGES - An Object Oriented UIMS. *Human-Computer Interaction - INTERACT'87*, edited by H.J. Bullinger and B. Shackel. 1987, 751 - 756.

[85] Solso, R.L. *Cognitive Psychology*. Allyn and Bacon, 1988.

[86] Stroustrup, B. Multiple Inheritance for C++. *Computing System*. Vol 2, No. 4, Fall 1989, 367 - 395.

[87] Stroustrup, Bjarne. *The C++ Programming Language*. Addison-Wesley, 1987.

[88] Summers, R.C. Local-area distributed systems. *IBM System Journal,* Vol 28, No.2, (1989), 227-240.

[89] Takala, T. Communication Mediator - A Structure for UIMS. *User Interface Management System*, edited by G.E. Pfaff, Springer-Verlag, (1983), 59-66.

[90] Tanner, P.P. Mulit-Thread Input. *Computer Graphics*. Vol.21, No.2, April 1987, 142 - 145.

[91] Tanner, P.P., MacKay, S.A., Stewart, D. A. and Wein, M. A Multitasking Switchboard Approach to User Interface Management. *SIGGRAPH' 86*, Vol.20 No.4, 1986, 241 - 248.

[92] Tesler, L. Programming Experiences. *BYTE.* August 1986, 195 - 206.

[93] Urlocker, Z. Object-oriented Programming for Windows. *BYTE.* May 1990, 287 - 294.

[94] Weber, H.R. Meditation on Man-machine Interfaces or Our Personal Role in Graphics Dialogue Programming. *Computer and Graphics.* Vol. 9, No. 3, 1985, 237 - 245.

[95] Welch, K.P. Using Object-Oriented Methodologies in Windows Applications. *Microsoft Systems Journal.* May 1990, 63 - 66.

[96] Wiener, R.S., Pinson, L.J. *An Introduction to Object-Oriented Programming and C++*. Addison-Wesley, 1988.

[97] Woelk, D., Kim, W., and Luther,W. An Object-Oriented Approach to Multimedia Databases. *In Proceedings of the ACM SIGMOD CONFERENCE (1986). ACM,* 311-325.

CUHK Libraries

000325481