Graphical Interface Services for Application Integration

Susan Franklin
Tony Peters

International Business Machines Corporation
5 West Kirkwood Drive
Roanoke, TX 76299

Abstract

This paper describes a set of graphical object services which provide a cohesive, consistent application development environment for implementing graphical user interfaces in intelligent workstation applications. These objects are implemented as a set of controls using the Presentation Manager™ windowing system. A control in the Presentation Manager environment is a user interface element with a unique programming interface and application function. This programming interface consists of a series of messages passed between the creating application and the control. The application environment as well as the operating environment of the Presentation Manager is described. The graphical object services are described functionally, as well as architecturally, in order to describe the capabilities of the objects and the necessary detail required to implement new objects. Protocols required to pass extensive amounts of graphical data efficiently between objects in a multi-tasking environment are discussed.

Keywords. User Interface, Graphical Programming Services, Windowing, Presentation Manager, Programming Tools, Application Integration, Office Automation, Icon, Direct Manipulation, Electronic Office.

Introduction

The incorporation of graphical user interfaces into computer applications has promoted the growth of advanced user interaction styles and techniques. As application developers attempt to implement many of these new concepts, the need for a consistent set of graphical user interface services for applications has surfaced. Some of this enhanced capability can be provided by graphical window-oriented operating systems. Other services must be provided by the application itself as requirements continue to change. These services must be implemented such that they are consistent within the application and, ideally, usable by other programs as well. The strategy used in providing graphical services for applications is to provide services as building blocks which a programmer may use to build the specific graphics-oriented task at hand. These building blocks, as well as the operating system services and communications protocols, may be used by various applications to achieve a graphical user interface that is not only intuitive, but also consistent among a set of integrated graphical applications.

Application Development Environment

The target environment in this discussion of graphical interface services is the IBM Operating System/2™ (OS/2™) with the OS/2 Presentation Manager user interface. OS/2 is a multi-tasking operating system for IBM or compatible personal computers. Application programs designed to execute on OS/2 and Presentation Manager may run concurrently with other OS/2 applications, while sharing many of the user interface concepts and interaction styles of the Presentation Manager. While a complete discussion of the OS/2 operating system concepts is beyond the scope of this paper, a brief introduction follows.

Basic Operating System Services

It is the goal of OS/2 to provide a platform for application development and execution which provides sufficient resources to multiple applications for completion of desired tasks. To achieve this goal, OS/2 provides support in resource management, multi-tasking, and application development tools.

In the OS/2 environment, multiple applications may run, sharing the same system resources. The operating system ensures that resources such as memory or files are accessed such that an application's data is protected from corruption by another application in the system. Shared resources such as processor time are parcelled out by OS/2 based on a system priority scheme. A program which accesses another application's data area is terminated by OS/2, causing the ill-behaved application to end, rather than bringing down the entire system.

Application development tools which support the design and development processes are also provided with OS/2.

The primary focus of this discussion is on the end-user interface toolkit, called the Presentation Manager.

User Interface Development with Presentation Manager

The Presentation Manager allows multiple OS/2 applications to appear on the screen at the same time. Each application displays its data in a rectangular, bordered area on the screen called a ***window***. An application may choose to display its data in more than one window. The Presentation Manager provides a set of user interface building blocks, such as a window title bar or a pulldown menu, that allow windows from different applications to behave in a similar manner. Common user actions such as sizing a window, moving a window, scrolling information, or selecting commands from menus are handled consistently by the Presentation Manager. The application-specific data is displayed in an area known as the ***client window*** which is the white space in the center of an application window. Figure 1 shows a typical application window. Each of the system provided building blocks is labelled.

Once an application displays its data in a window, it receives messages from the system regarding user input to the window. For example, if data is entered into a window or the window is resized, the system notifies the application's window. The window's response to these messages is what determines the path of execution for the program.

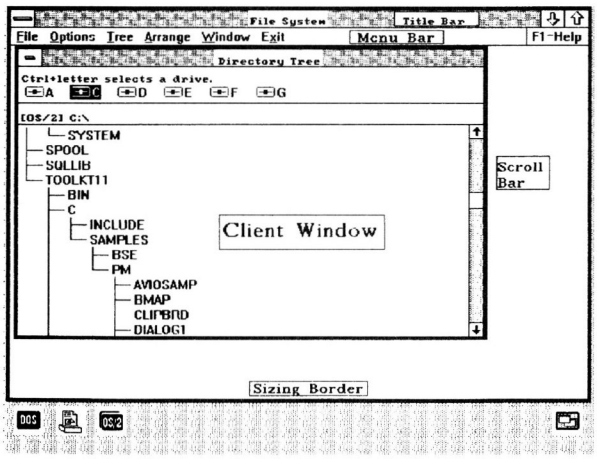

Figure 1. Standard Presentation Manager Application Window

Structure of a Presentation Manager Application

A Presentation Manager application always consists of a main initialization procedure and at least one window procedure. The main procedure initializes the OS/2 resources used by the program, sets up a ***message queue*** to receive the system messages, and begins a message processing loop to get messages from the queue and dispatch them to the appropriate window. The window procedure is the code responsible for receiving and processing the messages.

The main procedures for all Presentation Manager applications are nearly identical. It is the window procedure which distinguishes the behavior of one application from another. A window procedure may be used for more than one window. For this reason, a window procedure is said to define a ***window class***. That is, each window created by an application using the same window procedure will behave in the same manner. An application may define as many window classes as necessary.

The primary responsibility of a window procedure is to respond to messages sent to a particular window. A window procedure may choose not to respond to a message in which case the system will perform some default action. A window procedure simply checks for messages that have a prescribed response and performs the desired action. Certain system messages are handled differently by each window class. A primary example is the ***WM_PAINT*** message, which is sent to a window by the system when the window needs to be repainted. Since each window class looks different, the response to the WM_PAINT message must be different.

Application Overview

The application environment for which the application integration graphical interface services are developed is that of a set of office applications. The versatility of these graphical application services is in no way limited to office applications, but with the office environment being our current development thrust, the majority of this paper will discuss the application integration graphical interface services within the context of the office application set.

This office environment is the electronic counterpart of a real office. It makes considerable use of graphics to represent objects that are available to the user in a real office. Familiar office objects are presented to the user as small graphic or pictorial representations known as ***icons***. For example, the electronic equivalents of filing cabinets, folders, documents, waste baskets, telephones, calculators, clocks, and other common office items are represented on the display screen by icons.

In this electronic office, an ***object*** is any entity that can be manipulated as a single unit, or can be conceptually regarded by the user as capable of having an independent existence. Each of the above mentioned office representations is an object that can have activities performed on that object.

The office interface will provide a user environment with graphic presentation of office objects and activities. The

office desktop itself is an OS/2 Presentation Manager window and is presented when the office is started from the file system or from an OS/2 command line. This window contains an initial set of icons which represent office objects. Associated with each object is an ***Object Handler***. The object handler provides the icon that represents the object and the program to handle user actions on the object. Object handlers also handle interactions with other object handlers. The user has the ability to perform direct manipulation of objects in the office application. Through the mouse interface, the user can select an object as represented by an icon and drag it to another object to accomplish an action. For example, to discard an object from a mail basket, the user can select the icon representing the mail item and drag it to the "shred" icon.

Common Graphical Object Services

It is a primary requirement of the office graphical interface services to minimize the office Presentation Manager application programming efforts and the associated skills level required to develop a best of breed Presentation Manager application. In providing the graphical interface services for minimizing the application effort for applications, it is important that the graphical interface services do not sacrifice the inherent capabilities of the Presentation Manager in achieving application development efficiency. Office Presentation Manager application graphical interface services must be able to provide for the adequate exploitation of Presentation Manager capabilities across the entire spectrum of office applications.

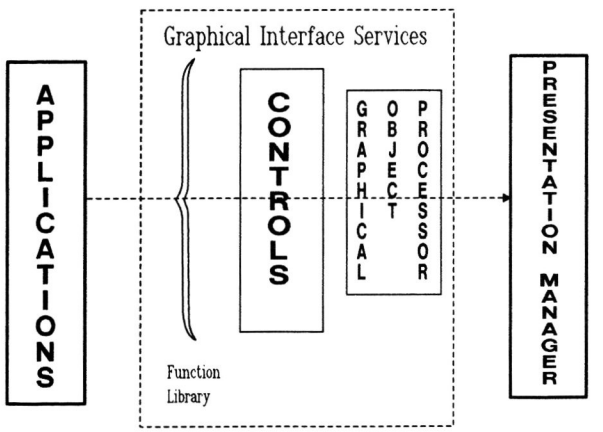

Figure 2. Graphical interface services functional flow

Another equally important objective of the graphical interface services is to provide for a tight interface consistency among all office applications. Additional goals of the office are to provide an open architecture of snap in applications and to facilitate a wide variety of application national language support.

The office common graphical object services approach to these requirements is to provide a Presentation Manager compatible architected set of common graphical object services. This is achieved by providing applications with a set of user interface building blocks, called ***controls***. The Presentation Manager common graphical object services are functionally illustrated in Figure 2.

User Interface Controls

A control in the OS/2 Presentation Manager environment is a user interface element with a unique programming interface and application function, such as a menu or scroll bar control. Typically, this interface element is a window. Examples of such window interface elements could be as simple as a graphic scroll bar or as complex as an icon control which displays and manages the direct manipulation of graphic icons. The programming interface to controls consists of a series of messages passed between the creating application and the control. The following sections describe how these controls are invoked by calling programs and how such controls are structured internally.

Invoking User Interface Object Controls

The section "Structure of a Presentation Manager Application" discussed the general program structure of a Presentation Manager program. The specific invocation of a user control will be discussed here in detail as well as the communication between the control and the owning application. The example control which will be used for this discussion will be an icon control which is a graphical control which lists its elements as icons. Objects may be inserted, deleted, or edited within this control. The icons may be rearranged though direct manipulation or through corresponding commands from the owning application. Figure 3 shows an owning application with an underlying icon control containing a list of several phone messages.

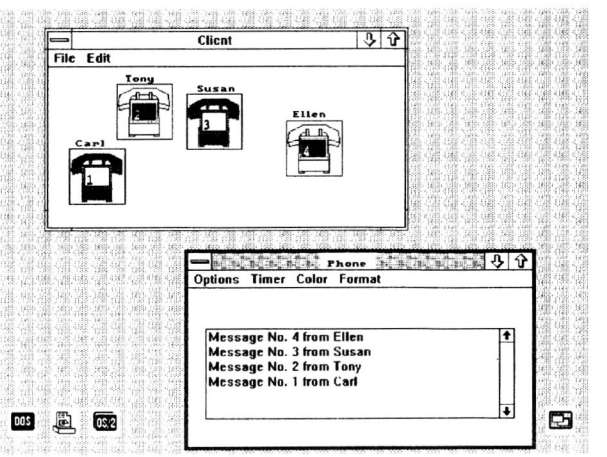

Figure 3. Example application with graphic user interface objects

The two primary responsibilities an application program has in creating a user control object is to ***register*** the control class and ***create*** the control. Each control is an instance of a particular type of window called a window class. Registration defines to the operating system all the necessary information required for an application to invoke the control. The external invocation identifiers, the processing routines and specific parameters such as storage requirements are specified in the control registration. The class style also defines the action that the operating system is to perform when moving or sizing operations occur on the screen and affect a window belonging to the registered class. Once a control has been defined to the operating system through registration the application can create as many controls of this specific class as it wishes.

Once the window class has been registered, the application creates the control by issuing a **WinCreateWindow** call of the registered class. Using our example of the icon control, the owning application issues the **WinCreateWindow** using the registered class of ***IconCtl***.

Four basic types of information must be passed when the control is invoked:

1. Owner information describing the ownership and parentage of the control. This information is necessary in defining the messaging matrix to the system for communication between the owning windows and the subordinate controls.
2. Size information during the creation of the control. This information can be omitted at the creation and can be dynamically supplied when the control is to be displayed.
3. Application specific parameters. In our ICON example, these parameters describe specifics of the icons within the control.
4. System required parameters such as a unique ID identifying the control to the operating system.

A unique handle is returned by the operating system from the window creation which is used by the application in communicating with the control.

Figure 4 shows the sample calling sequence for invoking the icon control.

The application and the control communicate by sending each other messages. Messages are addressed to the window handle which is returned when the windows are created ("User Interface Controls"). The message components are a message definition and two message parameters. The message definition indicates which message is being sent and the two parameters are data which is passed to the receiving window. The data parameters can be as simple as two or four byte fields or intrinsically complex as the passing of graphical iconic data. Specifically, how graphical data is passed is described in detail in "DDE Implementation" and illustrated in Figure 6. Messages can be sent synchronously and posted asynchronously.

```
WinRegisterClass((HAB)NULL,(PSZ)"IconCtl",
        (PFNWP)IconCtlWndProc,CS_SIZEREDRAW,4);
hCtl = WinCreateWindow(hwndParent,
                       (PSZ)"IconCtl",
                       (PSZ)NULL,
                       style,
                       x,y,cx,cy,
                       hwndOwner,
                       HWND_TOP,
                       id,
                       (PVOID)NULL,
                       (PVOID)NULL);
```

Figure 4. Sample icon control invocation by calling application

Graphic Object Control Implementation

While the implementation of a specific Presentation Manager user interface control will vary depending on the requirements, general guidelines exist for the implementation of a control. Successful development of a control begins by understanding the general characteristics and the skeletal structure of a control. This template can then be expanded to include the specific functional requirements of the control.

General Control Requirements

Message Handling: From an implementation viewpoint, a control is nothing more than a specialized window class which is expected to field certain messages and return the expected values. The internal structure of a control is simply a window procedure. There is no main program or invoking routine, since these tasks are performed by the application creating the control. The functionality of a control is determined simply by the kinds of messages that are accepted.

A control generally accepts two types of messages: general window messages predefined in the Presentation Manager and new control messages defined by the control. A subset of the former group is fielded by all controls. For example, the control must always be prepared to repaint its contents when the WM_PAINT message is received. Likewise, the control may need to reposition or resize its contents when the WM_SIZE message is encountered. WM_CREATE processing gives the control a chance to initialize data and set up storage blocks, while the WM_DESTROY message is the appropriate time to release all resources allocated for the control. Beyond these four messages, the specific purpose of the control determines what additional system window messages (such as WM_BUTTON1DOWN) or specific control defined messages must be fielded.

Control Parent and Owner Relationships: In addition to fielding certain messages and returning values, controls often post messages when certain specified events occur.

An icon control posts a notification message when the user selects an icon by clicking on it with the mouse. Notification messages such as these are posted to a window known as the control's owner. The owner is specified during creation of the control. The only logical relationship between a control and its owner is the fact that notification messages are posted to the owner. It is the parent of the window which determines creation, positioning, and destruction of the control. The parent and owner may be the same window, but do not necessarily have to be. For example, if an application wishes to place a control inside a dialog box, then the dialog box is the parent of the control. However, the standard dialog box is not set up to field notification messages from certain controls, so the owner of the control will be another window, possibly the main client window procedure of the application. The implementation of a user interface control must not confuse the parent and owner relationship. A simple rule is that all outgoing messages from a control are posted to the owner. A control should never have any reason to communicate with its parent.

Control Status and Instance Data: Since a user interface control is a resource made available to all applications, a control cannot make any assumptions regarding its origin. For example, the control may be invoked several times by the same application or by many different applications. Associated with each invocation of a control is a particular state. For example, each icon control contains a certain number of icons positioned in particular locations within the control. Data which describes such information is called ***instance data*** and must be stored such that each individual instance of a control can easily access its state data at all times during execution of the control.

Storage of instance data is accomplished through the use of a Presentation Manager concept known as ***window words***. During class registration, an application may specify a specific amount of data to be reserved for each instance of the window class. These data blocks, or window words, are accessible through a standard Presentation Manager API call. Given any window handle, the Presentation Manager can return a pointer to any of the requested window words. In order to maintain and access instance data, a control's window class must be registered with an additional four bytes of data. This window word will contain a long pointer to the control's instance data block, and may be accessed at any time during the control's execution. Since one of the parameters to a window procedure is the handle of the window receiving the message, there is no confusion as to which invocation of the control is executing, and the appropriate handle may be used to access the pointer to the instance data block.

Under this design approach, each user interface control has a standard four byte window word used to point to the instance data block. The size and contents of the instance data block will vary widely in different controls. In the icon control, this block contains information concerning the number of icons present, the size of the icons, the presentation format of the icons, as well as pointers to the data concerning each individual icon. The contents of the instance data block are likely to change as the developer iterates on the implementation of a user interface control. These iterations have no effect on any other applications, since the instance block pointer remains the same.

Enhanced Graphical Object Services: Once the basic internal structure and data blocks for the control are determined, the function provided by the control may be expanded. While the purpose and presentation style of graphical controls varies greatly, a common requirement of these controls is the manipulation of graphical objects. This includes the creation, insertion, and deletion of graphical objects in the control, as well as the selection, highlighting, and direct manipulation of the objects. A pulldown menu and an icon control may serve entirely different purposes, but both deal with the user selection of individual primitives which may contain graphics. If each implementation of a graphical control employs a different strategy for the manipulation of graphical objects, this could be reflected in the user interaction with the control, resulting in confusion. By providing a lower level of services for controls to be used in the manipulation of graphical objects, greater consistency can be achieved.

Primitive Graphical Objects: The Graphical Object Processor (GOP) is a set of common services that isolates the manipulation of primitive graphical objects from the control. Generally, a control keeps two kinds of information about its graphical objects. First, it keeps an internal data structure which describes the object. For example, the icon control creates a data block for each icon, describing the format of the graphics, the string name of the icon, and the ID value for the icon. Second, it keeps the actual picture of the icon. The goal of the GOP is take over management of the actual picture, leaving the data block to the discretion of the control.

Management of the actual graphics is performed by the GOP using the Presentation Manager GPI interface. Each graphic object is stored as a GPI segment, allowing repositioning and user selection through correlation. Each graphical segment is kept in a specified position on a segment chain. The control is responsible only for providing the GOP with the source graphics in any of the acceptable input formats (drawing orders, bitmaps, or metafiles). The GOP will in turn create the segment, draw any borders or add any text which the control has defined, and position the picture. If the user selects one of the graphical objects, the GOP will detect selection and highlight the object in any manner that the control has defined. If the control allows direct manipulation of the graphical objects, the

GOP will detect the mouse movements and perform the tracking of the object across the screen. It will reposition the graphical object when tracking is complete.

By providing additional graphical object services to controls, the controls can be masked from the segment manipulation, graphical transforms, and coordinate conversion that is necessary to create, display, and manipulate graphical objects. Different input formats will be handled so that the control need not know the original format of the objects. Instead, the control may focus on the relationships between these objects and the notifications necessary upon the manipulation of the objects.

Graphical Inter-Program Communication

Throughout the office applications, it will be necessary to display graphical representations of office objects produced by many different applications. These objects will be displayed in a variety of formats, usually by a graphical list control on behalf of an application which has produced a list of office objects.

Office graphics management has been designed to manage the exchange of graphical data between the applications which create the data and the list control that will ultimately display some representation of the data. This exchange must allow the applications to create, update, and maintain their graphical data independently of the associated list control, while providing the application which generated the list with a copy of the graphical data and informing it when updates occur.

Many portions of the office user interface make use of iconic representations of both applications and data objects. Such iconic representations will occur both within windows and various forms of lists. The user can perform various actions on icons such as clicking an iconic object to expand it into a window or dragging an icon into another window. An example of such an action is dragging an iconic representation of a document into the main office window, placing the document icon on top a printer icon in order to print a document. Another such example would be to select a document which is in an iconic state, expanding it to its full textual form.

Icons can be constructed in two ways. The first is as a bit image. The second way an icon can be constructed is by using the OS/2 Presentation Manager GPI graphics API. An application would typically use a bit image icon for a completely static representation of an object. That is, if an object when it is in an iconic state is never going to change its appearance, then that object is a good candidate for using a bit image type icon. However, if an application will be using an icon dynamically to indicate its current state, then that application should probably be using the OS/2 Presentation Manager drawing orders to construct its icons. A good example is the mail application. The icon is to be updated each time a piece of electronic mail arrives, perhaps displaying a count of the items in the inbasket. In this example, the mail application would change both the content and count within the icon.

Office graphics management supports both bit image and presentation graphics forms of icons. Applications can send both forms of icons to office windows by storing the bit images or the graphical representation of the icon in a segment of shared memory and making that shared memory available to the receiving application. Figure 5 illustrates how graphics data is shipped between applications.

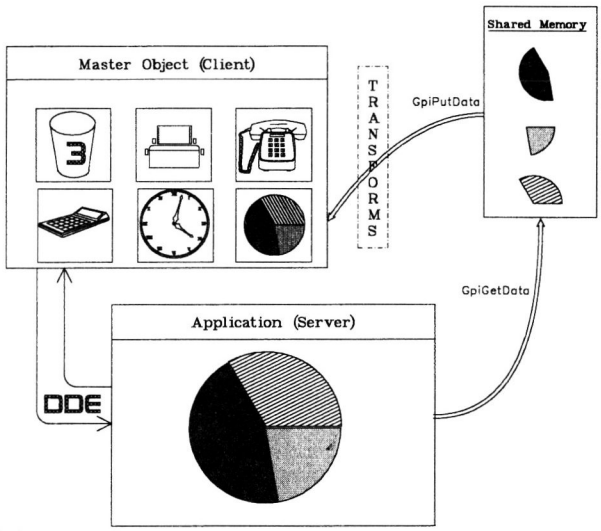

Figure 5. Graphics Data Flow

The communications between office applications exchanging graphics will be achieved through the use of the Presentation Manager Dynamic Data Exchange (DDE) protocol. Parameter passing conventions for the Presentation Manager version of DDE have changed significantly from the original Microsoft® Windows DDE protocol.

Office Graphical Data Exchange Model

In the office applications, the collection of available office objects in the system, as well as the graphical representation of those objects, is constantly changing based upon user actions. Those object handlers which choose to provide a dynamically descriptive icon must communicate with any graphical list control which is displaying a representation of that object. Generally, the object handlers which are providing the graphics are the "server" applications in the DDE model, while those applications receiving the graphics are the "client" applications. Applications which intend to

communicate can make no assumptions concerning the identity of interested participants. Therefore, it is the responsibility of a newly activated client or server application to initiate a DDE conversation.

A server application is typically initiated when a new office object is opened by the user. If the application wishes to provide a dynamically descriptive icon, it must initiate a DDE conversation in order to provide any applications which display that icon with updates to the graphical data.

DDE Protocol Structure with GDE Blocks

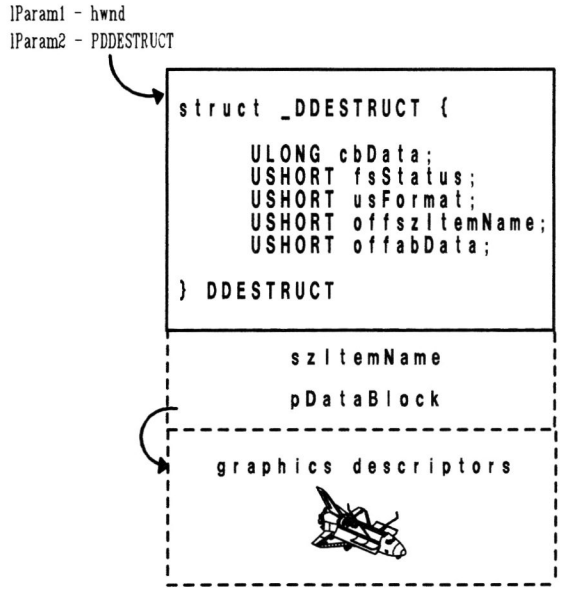

Figure 6. DDE message contents

User actions may also result in the addition of client applications, since there may be several graphical lists of office objects on the screen at one time, each continuously being updated by the applications managing these objects. These lists may or may not contain some of the same objects. When a new graphical list is created through some user action, the list may need to initiate conversations with server applications.

Since the number of client and server applications changes dynamically, a newly activated client (graphical list) or a newly activated server (object handler) must broadcast a conversation initiation message to all windows, to allow any interested applications to participate in the exchange.

DDE Implementation

The office implementation of the DDE protocol for graphical data exchange takes a two-tiered approach in the data sharing. The first layer of this implementation is the DDE messages and structures. Conversation initiation, update notifications, and data requests are transmitted using the DDE messages. In the majority of these messages, a pointer to a data structure, DDESTRUCT, is passed. This data structure contains two kinds of data: a fixed portion that is filled by all applications participating in conversations and a variable portion that differs from conversation to conversation. The first part of the structure contains information about the types of data being passed, the data formats, and the topic of conversation. These fields are provided in all DDE conversations. The second variable part of the structure contains the actual data being passed. In the case of office graphical data exchange, this data contains pointers to the second layer, the GDE control blocks.

Graphical Data Exchange Control Block

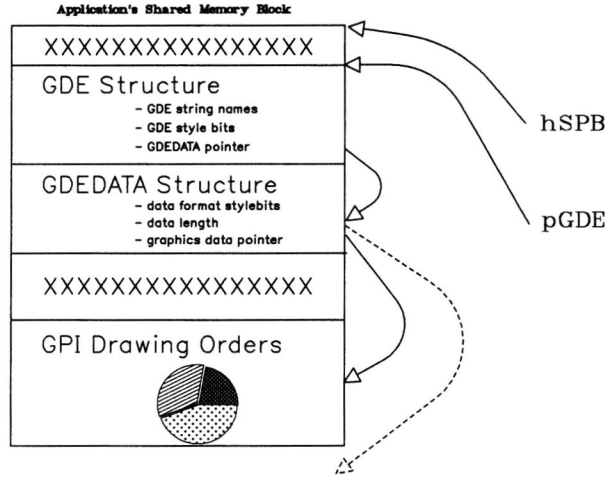

Figure 7. GDE Control Blocks

Figure 6 illustrates the relationship between the DDE messages and the GDE control blocks.

The second layer of the graphical data exchange implementation contains the GDE control blocks. The GDE data structure is used to identify the types of graphics being exchanged, the identity and class of the object handler providing the graphics, and other conversation specific information. The GDEDATA structure contains the actual graphical information. There may be multiple GDEDATA structures chained together. These GDEDATA structures can be passed directly by the receiving application to the graphical list control which will display the graphical objects.

Figure 7 illustrates the content and relationship between the GDE control blocks.

Conclusion

The development of user interface controls in order to implement newly defined user interaction techniques allows many different applications to provide new function in a consistent manner. System services and protocols also serve to link applications such that they share actual data as well as the presentation style. The strategy of providing these services as building blocks to be shared by applications rather than using all resource to develop a single, complex application ensures an increased level of integration among participating applications.

Bibliography

1. S. Franklin and T. Peters, "A Technical Study of Dynamic Data Exchange Under Presentation Manager,"*Microsoft Systems Journal*, Vol. 4, No. 3 (May 1989).
2. E. Iacobucci, *OS/2 Programmer's Guide*, McGraw-Hill, Inc., Berkeley, California (1988).
3. "IBM Operating System/2,"*IBM Personal System/2 Seminar Proceedings*, Vol. 5, No. 5 (May 1987).
4. *IBM Operating System/2 Internals Volume 2: Presentation Manager*, IBM International Technical Support Center, Boca Raton, Florida (1988).
5. *IBM Operating System/2 Version 1.1 Technical Reference Programming Reference: Volume 1*, IBM Corporation, Armonk, New York (1988).
6. *IBM Operating System/2 Version 1.1 Technical Reference Programming Reference: Volume 2*, IBM Corporation, Armonk, New York (1988).
7. *IBM Operating System/2 Version 1.1 Technical Reference Programming Reference: Volume 3*, IBM Corporation, Armonk, New York (1988).
8. "IBM OS/2 Standard Edition Version 1.1, IBM Operating System/2 Update, Presentation Manager (Part 1)," *IBM Personal System/2 Seminar Proceedings*, Vol. 6, No. 1 (April 1988).
9. "IBM OS/2 Standard Edition Version 1.1, Presentation Manager (Part 2)," *IBM Personal System/2 Seminar Proceedings*, Vol. 6, No. 2 (April 1988).
10. *Operating System/2 Version 1.1 Programming Guide*, IBM Corporation, Armonk, New York (1988).
11. *Operating System/2 Version 1.1 Programming Overview*, IBM Corporation, Armonk, New York (1988).
12. C. Petzold, "The Graphics Programming Interface: A Guide to OS/2 Presentation Spaces," *Microsoft Systems Journal*, Vol. 3, No. 3 (May 1988).
13. C. Petzold, "OS/2 Graphics Programming Interface: An Introduction to Coordinate Spaces," *Microsoft Systems Journal*, Vol. 3, No. 4 (July 1988).
14. C. Petzold, *Programming the OS/2 Presentation Manager*, Microsoft Press, Redmond, Washington (1988).
15. K. Welch, "Creating User-Defined Controls for Your Own Windows Applications," *Microsoft Systems Journal*, Vol. 3, No. 4 (July 1988).
16. K. Welch, "Inter-Program Communication Using Windows' Dynamic Data Exchange," *Microsoft Systems Journal*, Vol. 2, No. 6 (November 1987).

Biography

Susan Franklin is a software developer for IBM Application Systems Division, Westlake, Texas. She is currently working in end user interface design and development for IBM office systems. She joined IBM in 1987. She graduated from Texas A&M University in College Station, Texas with a BS Degree in Computer Science.

Tony Peters is a software developer for IBM Application Systems Division, Westlake, Texas. His current assignment is development of the end user interface for the IBM office system applications. Previous assignments include both application and systems programming assignments. He joined IBM in 1982. Prior to joining IBM he was a Member of the Technical Staff at Bell Telephone Laboratories in Naperville, Il. He graduated from the University of Tennessee with a MS Degree in Computer Science.

Trademarks