EXHIBIT F

US005969705A

United States Patent [19]

Fisher et al.

[11] Patent Number: 5,969,705

[45] Date of Patent: Oct. 19, 1999

[54] **MESSAGE PROTOCOL FOR CONTROLLING A USER INTERFACE FROM AN INACTIVE APPLICATION PROGRAM**

[75] Inventors: **Stephen Fisher**, Menlo Park; **Eric Mathew Trehus**, Milpitas, both of Calif.

[73] Assignee: **Apple Computer, Inc.**, Cupertino, Calif.

[21] Appl. No.: **08/816,492**

[22] Filed: **Mar. 13, 1997**

Related U.S. Application Data

[63] Continuation of application No. 08/312,437, Sep. 26, 1994, abandoned, which is a continuation of application No. 08/084,288, Jun. 28, 1993, abandoned.

[51] **Int. Cl.**[6] **G09G 5/00**

[52] **U.S. Cl.** **345/114**; 345/345

[58] **Field of Search** 345/119, 120, 345/118, 113, 114, 115, 343, 344, 345, 346, 347, 348; 395/155, 156, 157, 158

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,313,113	1/1982	Thornburg .
4,484,302	11/1984	Cason et al. .
4,555,775	11/1985	Pike .
4,688,167	8/1987	Agarwal .
4,698,624	10/1987	Barker et al. .

(List continued on next page.)

OTHER PUBLICATIONS

"Notebook Tabs as Target Location for Drag/Drop Operations",IB, vol. 35, No. 7, Dec. 1992.

Microsoft Corporation, "Microsoft Windows Paint User's Guide," Version 2.0, 1987, pp. 8–10, 44–45.

Microsoft Corporation, "Microsoft Windows Write User's Guide," Version 2.0, 1987, pp. 60–65.

Microsoft Corporation, "Microsoft Word: Using Microsoft Word", Version 5 0, 1989, pp. 69, 88–93.

Screen Dumps from Microsoft Windows V 3.1, Microsoft Corporation 1985–1992 (14 pages).

WordPerfect for Windows V 5.1, WordPerfect Corporation, 1991 (16 pages).

Jeffrey M. Richter, "Implementing Drag–and–Drop," *Windows 3.1: A Developer's Guide*, 2nd Edition, M&T Books, A Division of M&T Publishing, Inc. (1992), pp. 541–577 (Chapter 9).

Charles Petzold, "Windows™ 3.1—Hello to TrueType™, OLE, and Easier DDE, Farewell to Real Mode," *Microsoft Systems Journal*, vol. 6, No. 5 Sep. 1991, pp. 17–26.

Jeffrey Richter, "Drop Everything: How to Make Your Application Accept and Source Drag–and–Drop Files," *Microsoft Systems Journal*, vol. 7, No. 3, May/Jun. 1992, pp. 19–30.

Future Enterprises Inc., A Microcomputer Education Course for: U.S. Department of Commerce "Studen Workbook for Quattro Pro 3.0—Concepts and Basic Uses," 1991 (3 pages).

Inside Macintosh, vol. VI, 1991, pp. 5–1 to 5–117.

Microsoft Windows 3.1, Step by Step, 1991, pp. 168–170.

Apple Computer, Inc., *Inside Macintosh, vol. VI* Table of Contents, 5–1 through 6–117 (1991).

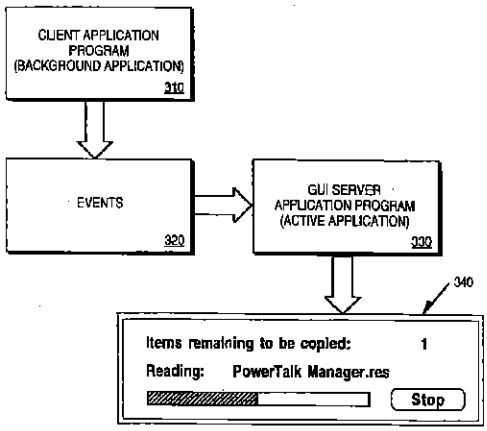
Primary Examiner—Chanh Nguyen

Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

Method and apparatus for a first process operative in a computer system controlling a user interface on a computer system display under control of a second process operative in the computer system. An event handler is installed for the second process, the event handler servicing events generated for controlling the user interface display under control of the second process. The first process may then perform a first set of functions in the computer system. The first process generates events for controlling the user interface display, the events related to the functions performed by the first process. The event handler receives the events generated by the first process and updates the user interface on the computer system display according to the events generated by the first process and received by the event handler.

1 Claim, 7 Drawing Sheets



U.S. PATENT DOCUMENTS

4,698,625	10/1987	McCaskill .	
4,720,703	1/1988	Schnarel, Jr. et al. .	
4,780,883	10/1988	O'Connor et al. .	
4,831,556	5/1989	Oono .	
4,862,376	8/1989	Ferriter et al. .	
4,868,765	9/1989	Diefendorff .	
4,905,185	2/1990	Sakai .	
4,922,414	5/1990	Holloway et al. .	
4,954,967	9/1990	Takahashi .	
5,047,930	9/1991	Martens et al. .	
5,079,695	1/1992	Dysart et al. .	
5,140,677	8/1992	Fleming et al. .	
5,157,763	10/1992	Peters et al. .	
5,196,838	3/1993	Meier et al. .	
5,202,828	4/1993	Vertelney et al. .	
5,214,756	5/1993	Franklin et al. .	
5,226,117	7/1993	Miklos .	
5,226,163	7/1993	Karsh et al. .	
5,228,123	7/1993	Heckel .	
5,260,697	11/1993	Barrett et al.	345/173
5,287,448	2/1994	Nicol et al. .	
5,301,268	4/1994	Takeda .	
5,305,435	4/1994	Bronson .	
5,333,256	7/1994	Green et al. .	
5,339,392	8/1994	Risberg et al. .	
5,341,293	8/1994	Vertelney et al. .	
5,371,844	12/1994	Andrew et al. .	
5,371,851	12/1994	Pieper et al. .	
5,400,057	3/1995	Yin .	
5,422,993	6/1995	Fleming .	
5,442,742	8/1995	Greyson et al. .	

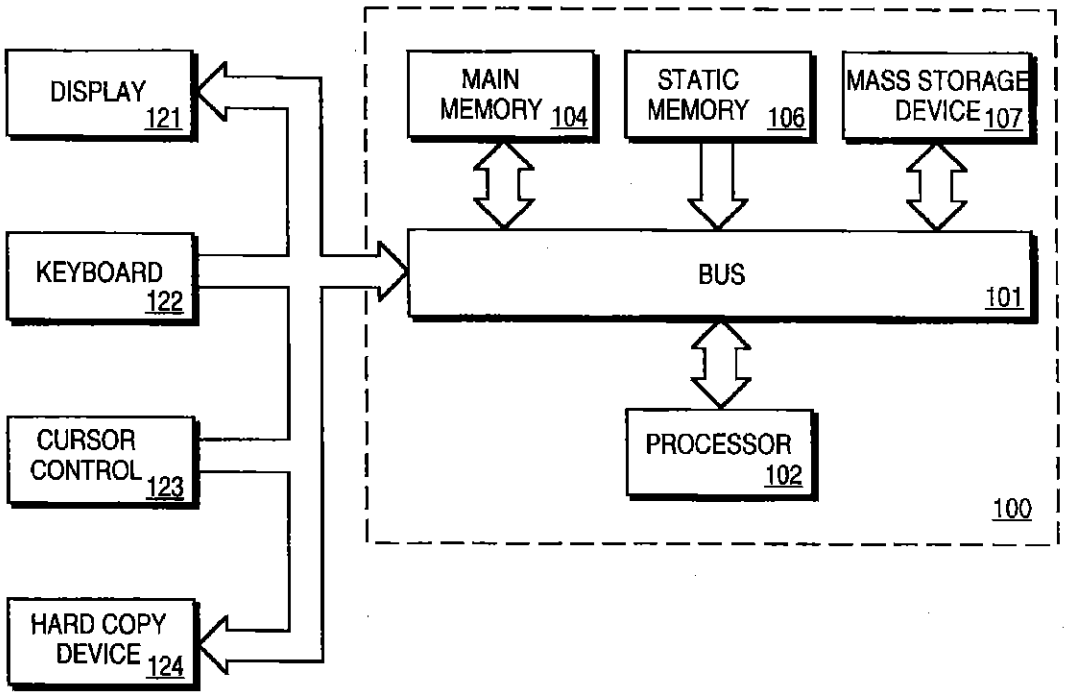

DISPLAY 121
KEYBOARD 122
CURSOR CONTROL 123
HARD COPY DEVICE 124
MAIN MEMORY 104
STATIC MEMORY 106
MASS STORAGE DEVICE 107
BUS 101
PROCESSOR 102
100

FIG. 1

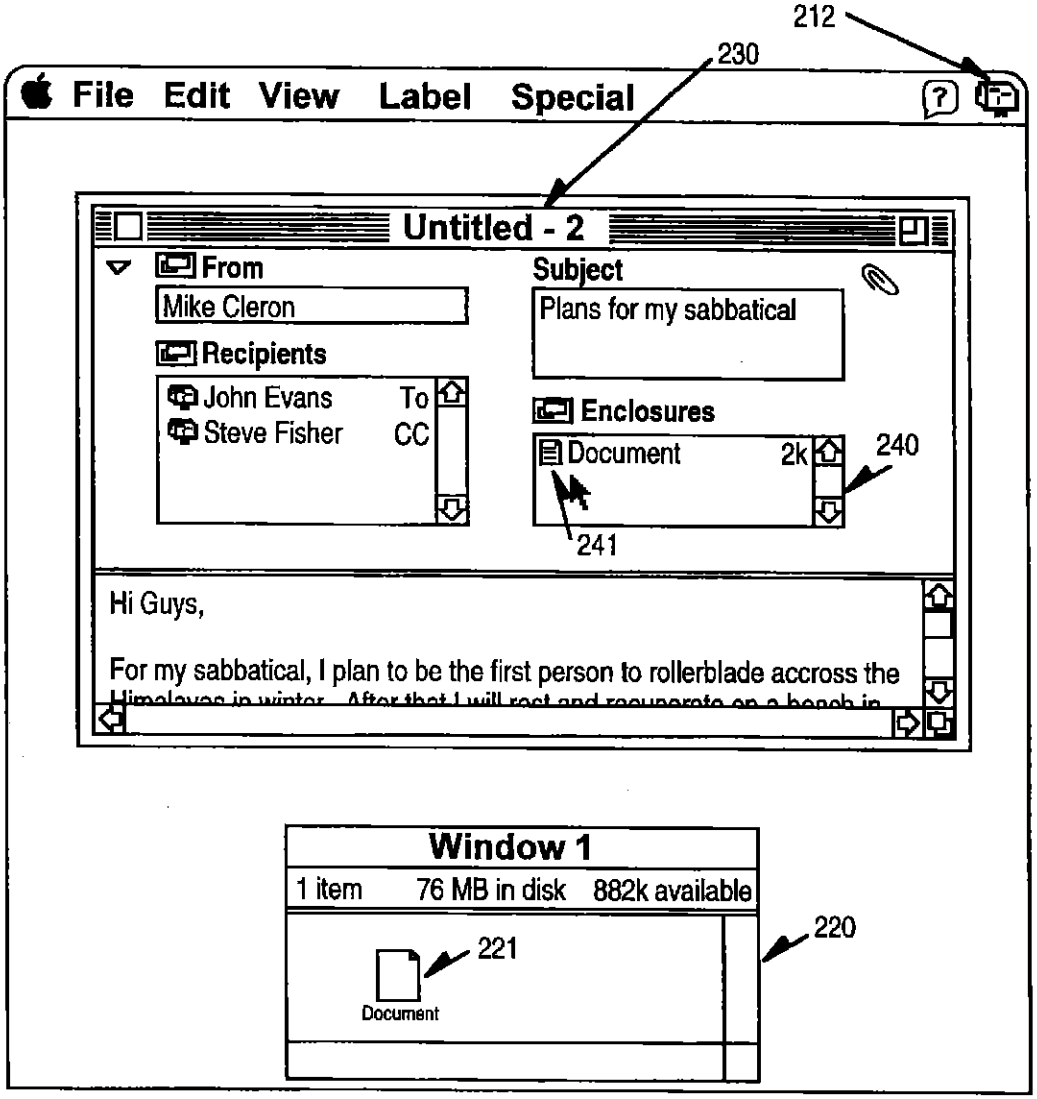


FIG. 2
(PRIOR ART)

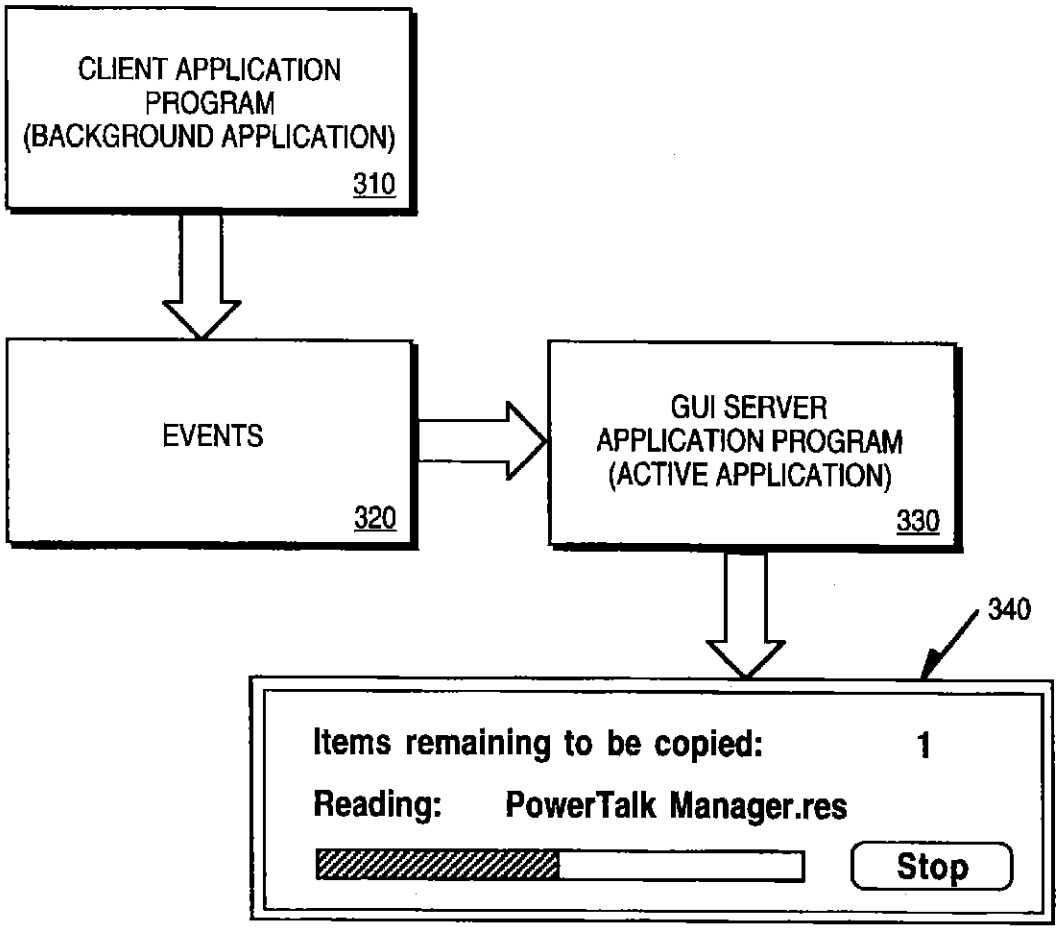

CLIENT APPLICATION PROGRAM (BACKGROUND APPLICATION)
310
EVENTS
320
GUI SERVER APPLICATION PROGRAM (ACTIVE APPLICATION)
330
340
Items remaining to be copied: 1
Reading: PowerTalk Manager.res
Stop

FIG. 3

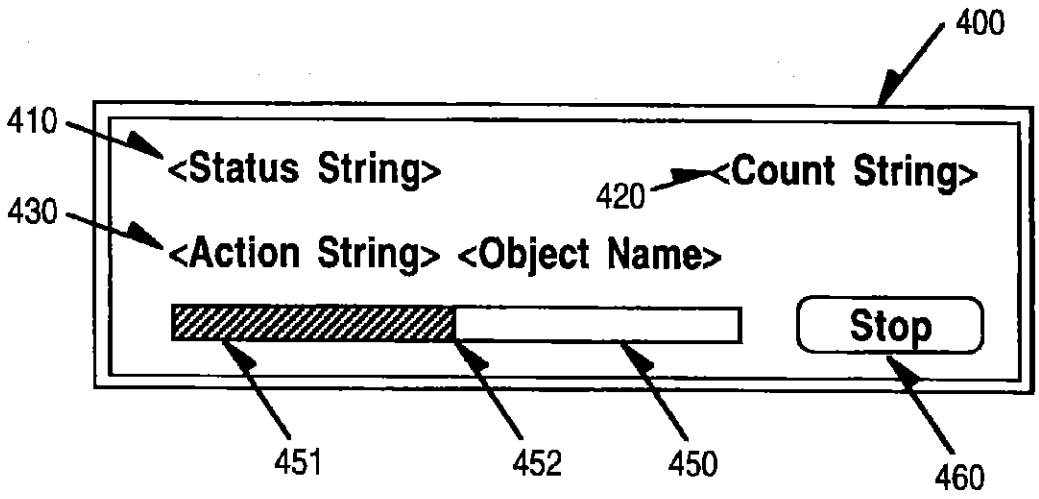


FIG. 4
(PRIOR ART)

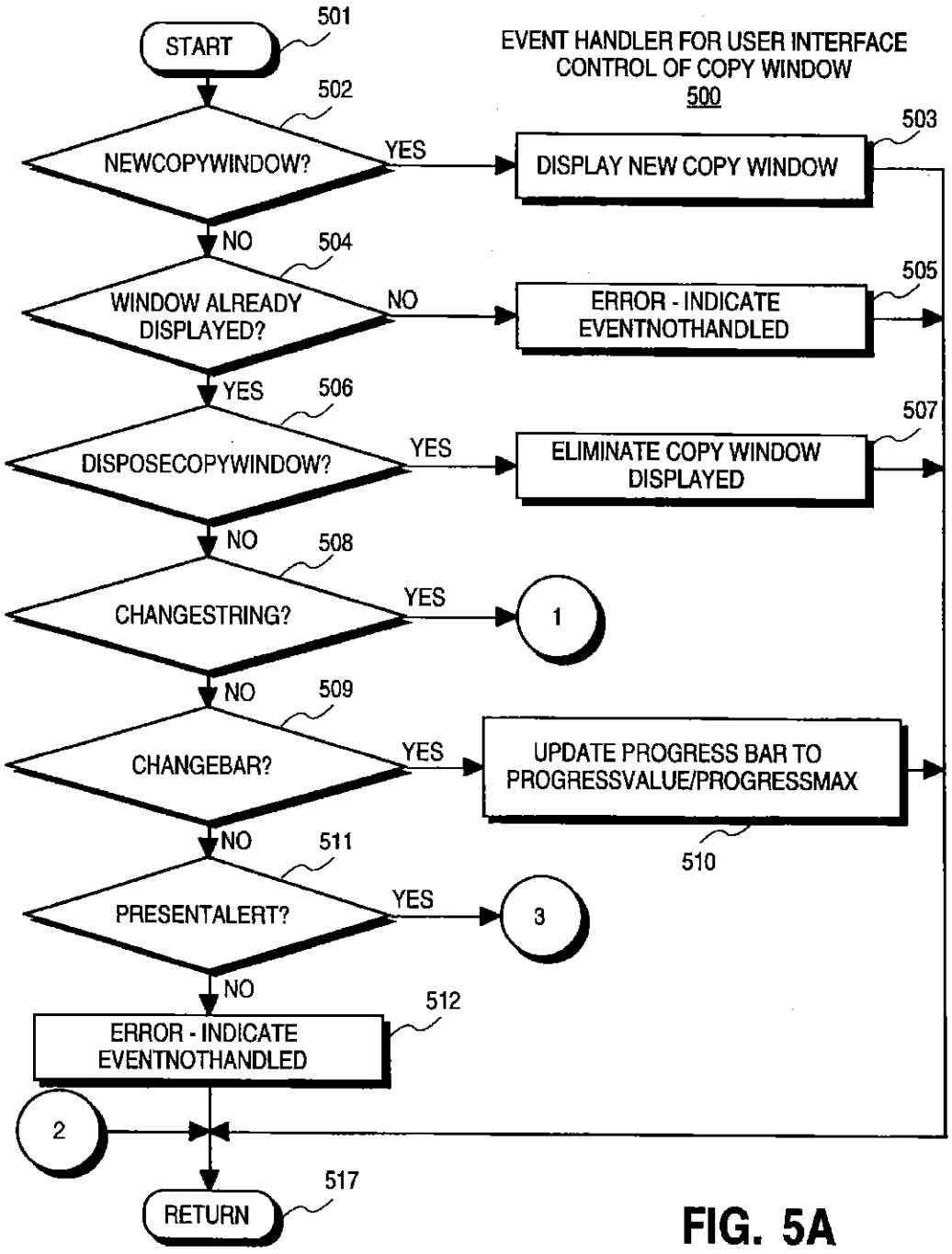

EVENT HANDLER FOR USER INTERFACE CONTROL OF COPY WINDOW
500
START
501
NEWCOPYWINDOW?
502
YES
DISPLAY NEW COPY WINDOW
503
NO
WINDOW ALREADY DISPLAYED?
504
NO
ERROR - INDICATE EVENTNOTHANDLED
505
YES
DISPOSECOPYWINDOW?
506
YES
ELIMINATE COPY WINDOW DISPLAYED
507
NO
CHANGESTRING?
508
YES
1
NO
CHANGEBAR?
509
YES
UPDATE PROGRESS BAR TO PROGRESSVALUE/PROGRESSMAX
510
NO
PRESENTALERT?
511
YES
3
NO
ERROR - INDICATE EVENTNOTHANDLED
512
2
RETURN
517

FIG. 5A

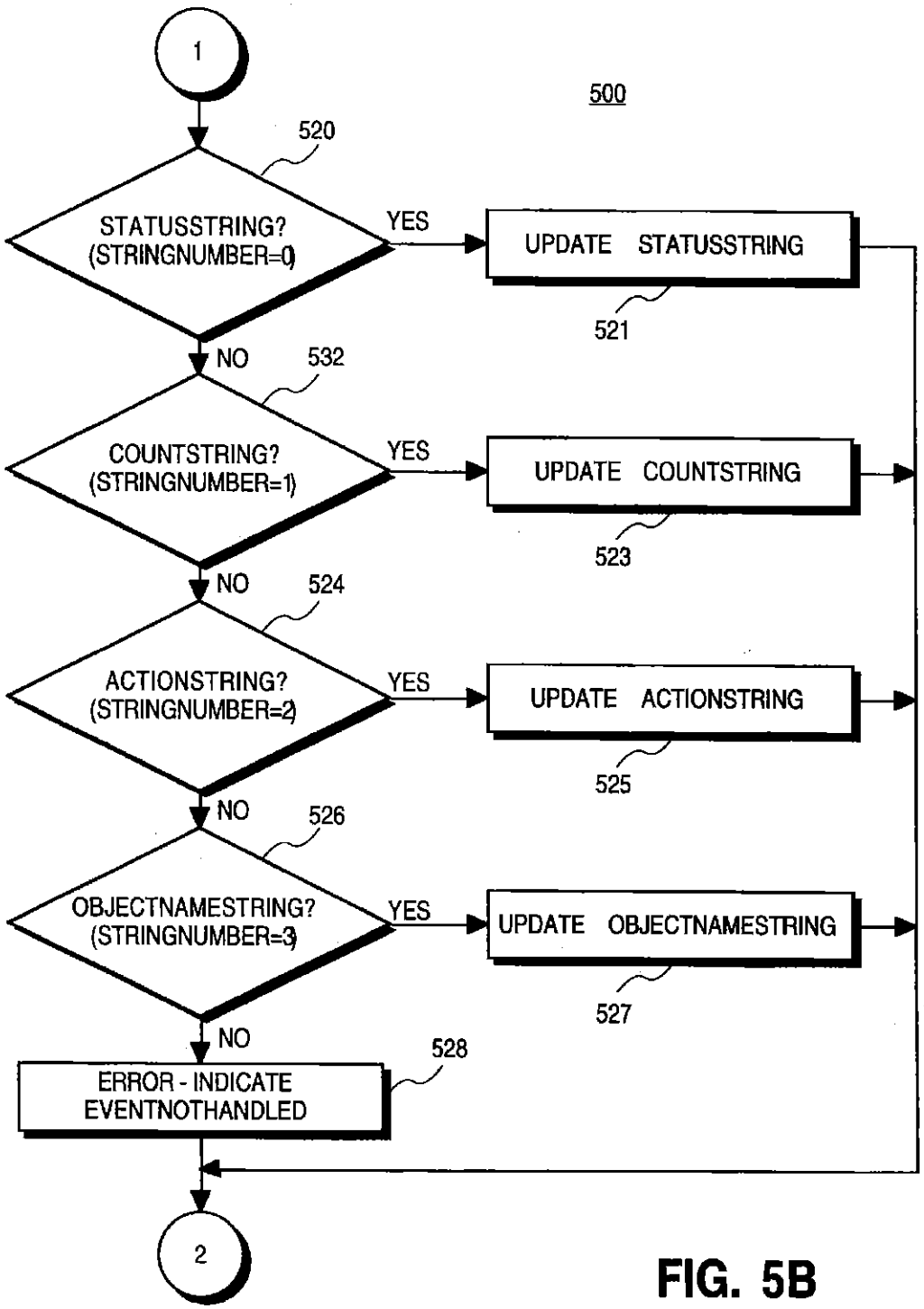

1
500
520
STATUSSTRING?
(STRINGNUMBER=0)
YES
UPDATE STATUSSTRING
521
NO
532
COUNTSTRING?
(STRINGNUMBER=1)
YES
UPDATE COUNTSTRING
523
NO
524
ACTIONSTRING?
(STRINGNUMBER=2)
YES
UPDATE ACTIONSTRING
525
NO
526
OBJECTNAMESTRING?
(STRINGNUMBER=3)
YES
UPDATE OBJECTNAMESTRING
527
NO
528
ERROR - INDICATE
EVENTNOTHANDLED
2

FIG. 5B

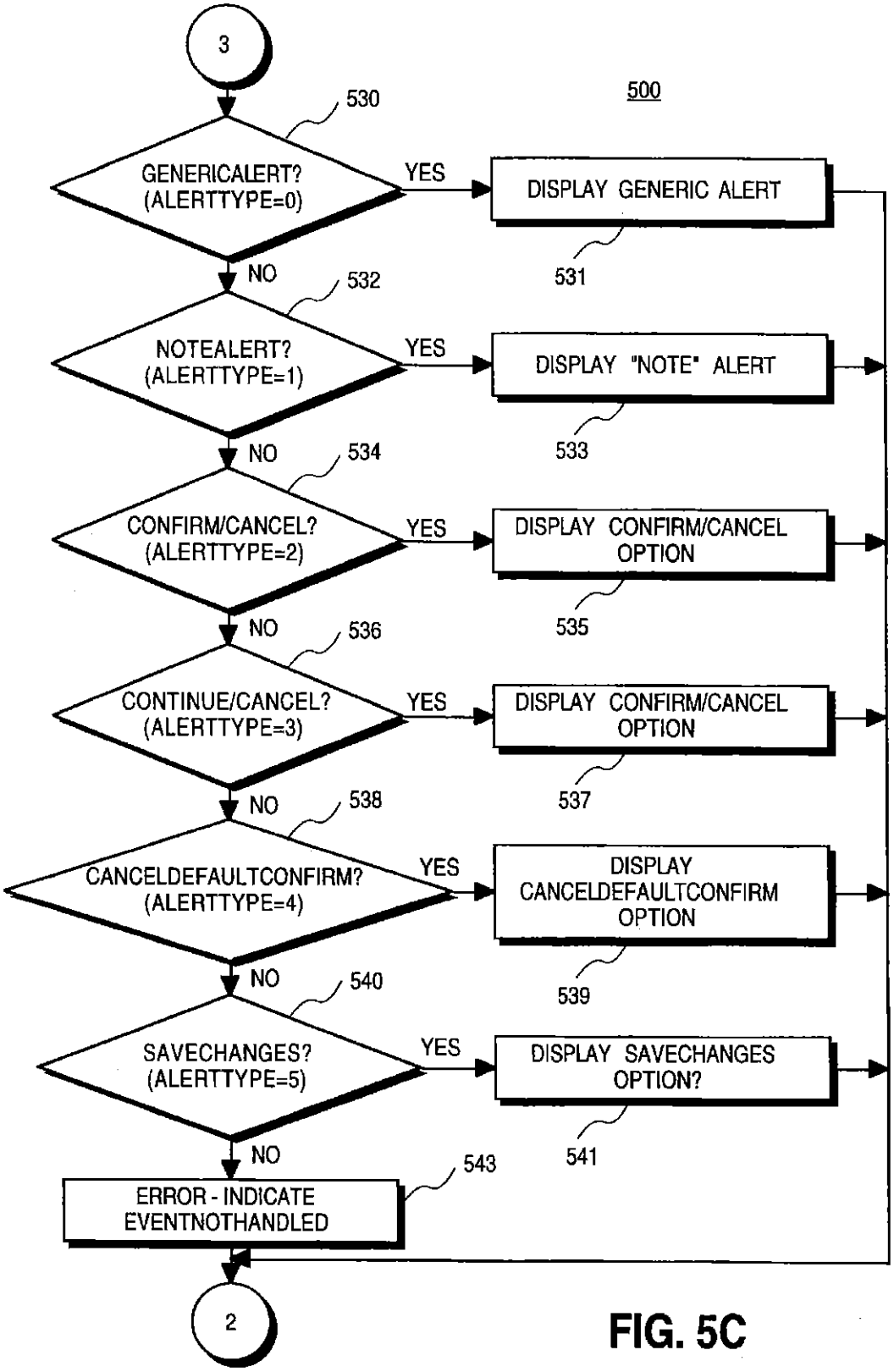

3
500
530
GENERICALERT?
(ALERTTYPE=0)
YES
DISPLAY GENERIC ALERT
531
NO
532
NOTEALERT?
(ALERTTYPE=1)
YES
DISPLAY "NOTE" ALERT
533
NO
534
CONFIRM/CANCEL?
(ALERTTYPE=2)
YES
DISPLAY CONFIRM/CANCEL OPTION
535
NO
536
CONTINUE/CANCEL?
(ALERTTYPE=3)
YES
DISPLAY CONFIRM/CANCEL OPTION
537
NO
538
CANCELDEFAULTCONFIRM?
(ALERTTYPE=4)
YES
DISPLAY CANCELDEFAULTCONFIRM OPTION
539
NO
540
SAVECHANGES?
(ALERTTYPE=5)
YES
DISPLAY SAVECHANGES OPTION?
541
NO
543
ERROR - INDICATE EVENTNOTHANDLED
2

FIG. 5C

MESSAGE PROTOCOL FOR CONTROLLING A USER INTERFACE FROM AN INACTIVE APPLICATION PROGRAM

This is a continuation of application Ser. No. 08/312,437, filed Sep. 26, 1994, now abandoned, which is a continuation of application Ser. No. 08/084,288, filed Jun. 28, 1993 status: abandoned.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to user interface control in a computer system. More specifically, the present invention relates to a messaging protocol which allows one application program to specify the appearance of an interface of a computer system while under control of a second application program.

2. Background Information

In multitasking operating systems, such as the Macintosh brand System 7.0 operating system available from Apple Computer, Inc. of Cupertino, Calif., typically, only one application program is given complete control of the user interface in order to prevent conflicts. There are circumstances, however, in which an application program which does not currently have control of the user interface will require that some information be presented to a user of the computer system. Typically, in the prior art, in these situations, the "inactive" application program must be "brought to the front" or made the active application (one in control of the user interface) in order for user interface control to become available to the application program. If events or other activities occur within the process that does not have control of the user interface, then the user may not be informed of the activity until after the activity has taken place, when the user brings the background application to the front. There are some circumstances in which the delay between the occurrence of the action within the background process, and the failure to provide feedback upon the computer system display may pose a substantial problem. For example, data may be overwritten, the user may wish to abort the task being performed, or he may wish to take corrective measures to otherwise address the activity occurring in the background task. There thus has arisen a need for background process to control the user interface which is currently under control of an "active" or foreground process within a computer system.

Another situation which frequently occurs is when one application program requires a complex service such as a file copying mechanism, but yet does not possess the necessary code in order to perform these tasks. An active application can use the services of the inactive application's processes without possessing the necessary code, and the inactive application program may drive the user interface of the "active" application program in order to provide feedback that the complex operation is taking place. Unfortunately, prior art techniques have no mechanism for allowing this to take place.

SUMMARY AND OBJECTS OF THE PRESENT INVENTION

One of the objects of the present invention is to allow a background application to provide user interface feedback when it is not the currently active application program.

Another of the objects of the present invention is to provide a protocol wherein a background application program may direct a foreground application program to control its user interface in a specified way.

Another of the objects of the present invention is to allow a background application program to communicate with a foreground application program for controlling the user interface of a computer system display.

Another object of the present invention is to allow a foreground application program controlling a user interface to take advantage of the services of a background application program.

Method and apparatus for a first process operative in a computer system controlling a user interface on a computer system display under control of a second process operative in the computer system. An event handler is installed for the second process, the event handler servicing events generated for controlling the user interface display under control of the second process. The first process may then perform a first set of functions in the computer system, in one embodiment, such as file management functions (e.g. copying and/or moving of files in the file system). The first process generates a first set of events for controlling the user interface display, the first set of events related to the first set of functions performed by the first process. For example, in various embodiments, feedback may be given about the progress of the file management functions (such as copying/moving specific files, reading from a source, and copying to a destination). The event handler receives the first set of events generated by the first process and updates the user interface on the computer system display according to the events generated by the first process and received by the event handler. This may include, showing the progress of the file management operation, and alerting the user of any abnormal conditions.

Other features, objects, and advantages of the present will become apparent from viewing the figures and the description below.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example and not limitation in the figures of the accompanying in which like references indicate like elements and in which:

FIG. 1 shows an example of a computer system architecture upon which one embodiment of the present invention may be implemented.

FIG. 2 shows a situation in which a file system manipulation may be performed when the filed management task is not the "active" application program (the program controlling the user interface).

FIG. 3 shows an event-driven architecture which is used in one embodiment of the present invention for allowing a background task to control a user interface of a foreground application program.

FIG. 4 shows an example of user interface display which may be controlled directly by a foreground application program or "server" process, but which may be directed by and whose functionality may be provided for by a background process (or "client").

FIGS. 5*a*–5*c* show process flow diagrams of an event handler which is registered for use by a server application program to service events generated by a client application program for user interface control.

DETAILED DESCRIPTION

The present invention relates to a messaging protocol between processes and a computer system wherein a first

process (e.g., a client process) sends messages to a second process (e.g., a server process) so that the client process can direct the appearance of the user interface under control of the server process. In this manner, the client process performs certain functions, and the server process controls all user interface functions such as the display of feedback for those functions. For the remainder of this application, various process steps, apparatus, data structures, message formats, events, parameters, and other information will be discussed in detail, however, these are merely for illustrative purposes and are not intended to limit the present invention. It can be appreciated by one skilled in the art that many departures and modifications may be made from these specific embodiments without departing from the overall spirit and scope of the present invention.

Referring to FIG. **1**, a system upon which one embodiment of the present invention is implemented is shown as **100**. **100** comprises a bus or other communication means **101** for communicating information, and a processing means **102** coupled with bus **101** for processing information. System **100** further comprises a random access memory (RAM) or other volatile storage device **104** (referred to as main memory), coupled to bus **101** for storing information and instructions to be executed by processor **102**. Main memory **104** also may be used for storing temporary variables or other intermediate information during execution of instructions by processor **102**. Computer system **100** also comprises a read only memory (ROM) and/or other static storage device **106** coupled to bus **101** for storing static information and instructions for processor **102**, and a data storage device **107** such as a magnetic disk or optical disk and its corresponding disk drive. Data storage device **107** is coupled to bus **101** for storing information and instructions. Computer system **100** may further be coupled to a display device **121**, such as a cathode ray tube (CRT) or liquid crystal display (LCD) coupled to bus **101** for displaying information to a computer user. An alphanumeric input device **122**, including alphanumeric and other keys, may also be coupled to bus **101** for communicating information and command selections to processor **102**. An additional user input device is cursor control **123**, such as a mouse, a trackball, stylus, or cursor direction keys, coupled to bus **101** for communicating direction information and command selections to processor **102**, and for controlling cursor movement on display **121**. Another device which may be coupled to bus **101** is hard copy device **124** which may be used for printing instructions, data, or other information on a medium such as paper, film, or similar types of media. Note, also, that any or all of the components of system **100** and associated hardware may be used in various embodiments, however, it can be appreciated that any configuration of the system may be used for various purposes as the user requires.

In one embodiment, system **100** is one of the Macintosh® family of personal computers such as the Macintosh® Quadra™ or Macintosh® Performa™ brand personal computers manufactured by Apple® Computer, Inc. of Cupertino, Calif. (Apple, Macintosh, Quadra, and Performa are trademarks of Apple Computer, Inc.). Processor **102** may be one of the 68000 family of microprocessors, such as the 68030 or 68040 manufactured by Motorola, Inc. of Schaumburg, Ill.

Note that the following discussion of various embodiments discussed herein will refer specifically to a series of routines which are generated in a high-level programming language (e.g., the C++ language available from Symantec of Cupertino, Calif.) and compiled, linked, and then run as object code in system **100** during run time. It can be appreciated by one skilled in the art, however, that the following methods and apparatus may be implemented in special purpose hardware devices, such as discrete logic devices, large scale integrated circuits (LSI's), application-specific integrated circuits (ASIC's), or other specialized hardware. The description here has equal application to apparatus having similar function.

Graphical User Interface

Before discussing the preferred embodiment in detail, a brief overview of the user interface used in this system is required. A "windowing" or graphical user interface (GUI) operating environment is used wherein selections are performed using a cursor control device such as **123** shown in FIG. **1**. Typically, an item is "selected" on a computer system display such as **121** using cursor control device **123** by positioning a cursor, or other indicator, on the screen over (or in proximity to) an object on the screen and by depressing a "selection" button which is typically mounted on or near the cursor control device. The object on the screen is often an icon which has an associated file or operation which the user desires to use in some manner. In order to launch a user application program, in some circumstances, the user merely selects an area on a computer display represented as an icon by "double clicking" the area on the screen. A "double click" selection is an operation comprising, while positioning the cursor over the desired object (e.g., an icon), two rapid activations of the selection device by the user. "Pull-down" or "pop-up" menus are also used in the preferred embodiment. A pull-down or pop-up menu is a selection which is accessible by depressing the selection button when the cursor is pointing at a location on a screen such as a menu bar (typically at the top of the display), and "dragging" (moving cursor control device **123** while the selection button is depressed) until the selection the user wishes to access is reached on the pull-down menu. An item is indicated as being "selected" on a pull-down menu when the item is highlighted or displayed in "reverse video" (white text on a black background). The selection is performed by the user releasing the selection device when the selection he wishes to make is highlighted. Also, in some GUI's, as is described in the background above, the "selection" and "dragging" of items is provided to move files about in the file system or perform other system functions. These techniques include "dragging and dropping" which comprises making a "selection" of an icon at a first location, "dragging" that item across the display to a second location, and "dropping" (e.g., releasing the selection device) the item at the second location. This may cause the movement of a file to a subdirectory represented by the second location.

Note also that GUI's may incorporate other selection devices, such as a stylus or "pen" which may be interactive with a display. Thus, a user may "select" regions (e.g., an icon) of the GUI on the display by touching the stylus against the display. In this instance, such displays may be touch or light-sensitive to detect where and when the selection occurs. Such devices may thus detect screen position and the selection as a single operation instead of the "point (i.e., position) and click (e.g., depress button)," as in a system incorporating a mouse or trackball. Such a system may also lack a keyboard such as **122** wherein the input of text is provided via the stylus as a writing instrument (like a pen) and the user handwritten text is interpreted using handwriting recognition techniques. These types of systems may also benefit from the improved manipulation and user feedback described herein.

One problem solved by the present invention is illustrated with reference to FIG. **2**. Window **200** of FIG. **2** illustrates

a typical Macintosh user interface display while one application program, such as an electronic mail program, controls the user interface display. This is illustrated by the icon present in the upper right-hand portion **212** of the display **200**. The operating system only allows a single application program to control the user interface at any given time. However, other application programs such as, those performing file and/or program management functions (e.g., the "Finder" within the Macintosh brand operating system), allow the launching of applications, programs, and the movement of files within the file system. In one embodiment of the present invention, a user may decide to "enclose" a file represented by icon **221** in the file system with the mail message represented on window **230**. The application program controlling window **230** does not possess file transfer or file transfer feedback capabilities, whereas the File System Manager (known as the "Finder" in the Macintosh) does possess these capabilities. Therefore, it is desired that the application program controlling window **230** have certain functions and user interface capabilities of the Finder. In a typical prior art systems, feedback is provided by the file management function to show that the file movement or copy operation is taking place. This typically takes the form of a progress bar on a typical prior art user interface to show the progression of the file transfer operation as it takes place in the file system. For example, if a plurality of files are moved in the file system, or copied from one media device to another, file names showing each transfer of each file, and a darkened representation is shown in the progress bar to show overall completion of the copying of the files is represented on the progress bar. This will be illustrated in more detail below.

Event-Driven Architecture

An operation such as copying or moving files from one location to another in the file system is a nontrivial task. In this embodiment, the application program controlling the user interface defers to the background task the "Finder" so that it may perform the file management functions for transfer and/or copy of the file(s). For example, several tasks need to be performed by the copy/movement process prior to copying or moving the files. For example, the destination subdirectory or other file location needs to be scanned to determine whether any of the transferred file names are equivalent to those already in the subdirectory. If so, the user needs to be alerted in order to determine whether he wishes to overwrite the existing files at that location or, perhaps, use a different name. In another instance, there may not be sufficient space at the destination to which the files are being moved for writing the files. In this instance, the user is alerted that there was not sufficient space on the storage medium to store the files, is informed that the operation was not successful, and any file(s) already written or other intermediate file information can be deleted. However, in a situation such as that illustrated in FIG. **2**, the underlying file management process cannot present this user feedback or alert information to the user because it does not presently have control of the user interface. The process having control of window **230**, an electronic mail application program, is currently in control of the user interface. Thus, an improved means for allowing the background process (e.g., the Finder) to control the user interface is used in various embodiments of this invention to present feedback to the user regarding the underlying functions that are taking place. This is performed via interprocess communication, in the Macintosh brand operating system, using Events and the accompanying operating system Event Manager and Apple Event Manager available from Apple Computer of Cupertino, Calif.

Interprocess communication is an important aspect of modern computer system design. For example, such inter-application communication has provided in modern computer systems, such as the Macintosh brand computer's operating System 7.0, available from Apple Computer of Cupertino Calif., through a mechanism known as the Event Manager. The Event Manager and the Apple Event Manager are used for handling a wide variety of functions within application programs such as detecting user actions—key clicks, selections using a cursor control device, the detection of the insertion of disks into a disk drive on the computer system, opening files, closing files, or other actions within the computer system. Typically, processes running within a Macintosh brand computer system comprise a main program loop known as an "event" loop which detect the occurrence of these events in the system. Then, the application typically branches to portions of the program to allow the event to be serviced. Such an event driven architecture forms the core of many application programs within many different types of computers and operating systems in present use but, in this embodiment, resides in a system such as **100** described above.

Various embodiments of the present invention use the Event Manager and the Apple Event Manager supplied by Apple Computer for interprocess communication between applications programs which are operative within computer system **100** during run time to implement the features described herein. The event driven architecture for message passing between a first application program (e.g., a client application program which is operative in the background), and a second application program (e.g., a server application program which is the active application controlling the user interface), is illustrated with reference to FIG. **3**. For example, using the event driven architecture specified in *Inside Macintosh, Volume* 6, pages 5–1 through 6–118, "client" application program **310** communicates with the Graphical User Interface (GUI) "server" application program or active application program **330** via events **320**. Each of these events are generated by the client application program **310** and are detected by the Apple Event Manager. GUI server application **330** registers a process with the Apple Event Manager known as an Event Handler so that whenever defined events are detected, the Apple Event Manager forwards the event(s) to the registered handler(s) and cause the handler(s) to be invoked and service the events. At that point, the handler may determine the appropriate action to take place. In various embodiments of the present invention described herein, the registered event handler for GUI server **330** will cause the activation and modification of a user interface display, in this case, copy window **340** illustrated in FIG. **3**. Upon the launching of the GUI server application program **330**, or any application program which may become an active application program during specified user actions (e.g., the copying of file(s), the server application program will register with the Apple Event Manager the handler(s) which are used to service the events, including those generated by any potential client application programs (e.g., **310** of FIG. **3**). In the situation where a defined event is not serviced by any handler which are registered by the application program, then, a default handler supplied by the operating system is instead used for servicing the events.

For the remainder of this application, in the embodiment discussed herein, it will be assumed that the "server" (e.g., **330** of FIG. **3**) has registered a handler which will service

user interface events. It will also be assumed that a client program (e.g., **310** of FIG. **3**) provides the underlying functionality for performing the actions represented by the user interface (e.g., copying files), which occurs upon the inactive program detecting that a file should be copied (or moved) from one directory to another, such as a directory for "Enclosures" within an electronic mail application program. This function may also be requested by server process **330** sending an event to a handler registered for client process **310**, such as "CopyFile" 'File 1' to 'Enclosures.'" The mechanics of this operation, however, will not be described in detail because they are beyond the scope of the present invention.

Client to Server Events for Controlling the User Interface

The following events are defined in this new protocol for communicating from client **310** to user interface server **330**:

1. NewCopyWindow;
2. DisposeCopyWindow;
3. ChangeString;
4. ChangeBar; and
5. PresentAlert.

As client application program **310**'s file copy operations progress, certain of these events are issued by client process **310** to server process **330**. Moreover, communication is provided via another set of events from server **330** to client **310** to indicate the success of the action indicated by the events, and user interface feedback in response to information presented on the user interface by server **330**. A description of each of the specific events and the parameters used in each of these events will now be discussed with reference to FIG. **4**.

400 of FIG. **4** illustrates a typical copy window which is displayed during a file copying operation well known in the prior art. However, each of the informative portions of the window are displayed with corresponding parameter name for the event in angled brackets (e.g., <status string> **410**, <action string> **430**, <count string> **420**, and <object name> **440**), which is replaced by the strings specified within parameters associated with the event(s) issued by client process **310**. The events and parameters associated with the events will now be discussed.

NewCopyWindow

The NewCopyWindow event is signaled by client application program **310** to indicate that a new copy window (e.g., **400** in FIG. **4**) should be displayed. Using typical prior art user interface commands, the server application program's handler creates upon the display screen a copy window **400** as is illustrated in FIG. **4** with the appropriate strings specified in the event parameters. Each of the event parameters for the NewCopyWindow are specified in the following order:

1. actionString (e.g., "reading"/"writing"/"verifying")
2. objectName (name of object being copied)
3. statusString (e.g., "Preparing To Copy," "Items remaining," etc.)
4. countString (how many items are being copied)
5. progressValue (an initial value, usually 0)
6. progressMax (a maximum value)

actionString is used for specifying the operation being performed. In typical prior art copy operations, the value thus is one of the following three strings: "Reading"; "Writing"; or "Verifying", for specifying the operation being performed. The specified action string is placed into region **430** of the copy window **400** upon detection of the NewCopyWindow event.

objectName is used for specifying the string which will appear at region **440** on copy window **400**. It is used for specifying the name of the object or file being copied. Feedback can thus provide to the user which object is currently in the process of being read, written, or verified.

statusString **410** is used to specify the intermediate status of the copy operation taking place. For example, in certain prior art systems, this string may read "Preparing to Copy," "Items Remaining," etc. The status string indicates to the user the current status of the copy operation taking place.

countString is used for specifying the value which is shown at region **420** of display **400**, such as the number of items (e.g., files or bytes) which are being copied. Thus, in one situation, this string may contain "120 bytes" when a file or file(s) of 120 bytes in length are being copied.

progressvalue and progressMax are used for specifying the status of progress bar **450**. For example, the progressMax parameter is used for specifying some maximum value at which the progress bar will be completely darken. In an instance where a total of 120 bytes are being copied, progressMax may be equal to an integer value, such as 120. The progressValue parameter will thus be used to specify an intermediate value from some initial value (in one embodiment, the integer 0) to the progressMax value. Thus, on the display, progress bar **450** may be filled with a darkened region **451** up to an intermediate position **452** based upon the fraction progressValue/progressMax. For example, if progressValue equals 60 and progressMax equals 120, then progress bar **450** will have a representation such as that shown in FIG. **4** wherein $^{60}/_{120}$ or ½ of the progress bar has been darkened. Feedback is thus provided to the user to illustrate the current completion of the copy operation. In typical situations, the progressvalue will have an initial value such as 0, and the progressMax value will be some nonzero value, for example, equivalent to an integer representing the maximum number of bytes to be copied.

DisposeCopyWindow

The DisposeCopyWindow event is used for indicating the termination of a copy operation. This event causes the server's handler to remove window **400** from the display using well known prior art interface techniques. The event has no parameters because it merely removes from the display the currently displayed progress bar window.

ChangeString

The ChangeString event has the following parameters:

stringNumber statusString = 0 countString = 1 actionString = 2 objectNameString = 3
stringValue

stringNumber—For each of the above specified values of stringNumber, the identified string modified in copy window **400** using the ChangeString event according to the string contained in stringValue. statusString **410**, countString **420**, actionString **430**, or objectNameString **440** may be modified within, copy window **400** illustrated in FIG. **4**. The client application may thus cause updates to be performed within copy window **400** so that the user is informed of the current status of the copy operation (such as a current file being copied)

stringValue is a string which will replace the string specified by the integer value contained in stringNumber.

ChangeBar

The ChangeBar event has the following parameters:

progressValue (a current value)

progressMax (a maximum value) Each of these parameters are integer values specifying the current progression and the maximum progression of progress bar **450** of copy window **400**, is discussed with reference to the NewCopyWindow event above. The progress bar is thus adjusted to have a filled in representation, such as that shown as **451** according to the fraction progressValue/progressMax. The progress bar is update by the server's handler using standard prior art user interface commands.

PresentAlert

The following parameters are defined for the PresentAlert event:

```
alertType
        GenericAlert = 0
        NoteAlert = 1
        Confirm/Cancel = 2
        Continue/Cancel = 3
        CancelDefaultConfirm = 4
        SaveChanges = 5
alertString
```

alertType is an integer value which is used for specifying the type of alert displayed which is displayed to the user. Note that these alerts are all similar to those which are displayed in typical prior art copy operations upon detection of certain conditions, abnormal or otherwise. Some of these specified alerts require that the user respond. For example, the Confirm/Cancel alert displays a window which requests that the user "confirm" or "cancel" an ongoing operation. For example, the Confirm/Cancel alert may be used when the same file name is detected at a destination directory for a file which is being copied. Any of the standard alert windows which may be used in certain prior art copy operations may be specified using the proper alertType integer value. User responses to alerts will be provided with events from user interface server **330** to client **310** via another set of events discussed below.

alertString is a string value indicating the associated message to be associated with the alert. For example, the client application program may determine that a file name having an equivalent file name to a file being copied already resides at the destination application. In this event, the alertString may contain a message such as "File 'My File' already exists. Replace?" In any event, using the foregoing alert parameters, the PresentAlert event may specify appropriate alerts to the user.

Server to Client Events

All of the above events are shown for illustrative purposes only and are for the client application process **310** (e.g., the "Finder" performing the copy operation) alerting server process **330** (e.g., an electronic mail application program) to change the user interface display in a specified manner. However, other events may also be defined to specify other changes to the user interface display and for other operations, especially in instances where user interface response(s) to alerts are required. Because the background process (e.g., client process **310**) cannot detect user interface actions (such as selections on the display), the handler for server process **330** must detect these actions and transmit response event messages to client process **310**. In this case, client **310** will have its own event handler registered for servicing events issued by server **330** to client **310**. For example, if the user wishes to "cancel" an operation, an event entitled "CopyCancel" may be issued to the client process **310** in one embodiment of the present invention so that any ongoing operation(s) may be aborted. This is detected by a user selecting "Stop" button **460** at any time during the operation. The cancel operation may be detected by server **330** by the detection of a "mouseDown" event at a specific location, such as "Stop" button **460**, or its keyboard equivalent (e.g., a command period combination in the Macintosh). In this case, client application **310**'s handler may be alerted that an abort was indicated and take appropriate action.

In another embodiment, responses to alerts such as file overwrite messages may be sent from server **330** to client **310**. In this instance, an integer may be passed as a parameter wherein one value of the integer (e.g , 0) causes the operation to be aborted or a second value (e.g., 1) causes the file overwrite to be confirmed. Responses to alerts are performed using other defined events from GUI server **330** to client **310**. The user may be presented with the option of either confirming replacement of the file or canceling the copy operation. In one embodiment, when an alert is presented, client **310** remains idle until a response is made by a user on the user interface display. Then a corresponding response event (e.g., AlertReply) with a response parameter (e.g., an integer value Result containing an integer 0 indicating confirmation of the operation or integer 1 indicating canceling of the operation) is generated by server **330** to client **310** to either confirm or cancel the operation being performed by client **310**.

Other user responses to queries, such as alerts, errors, or other conditions, may also be responded to in this manner, as detected by GUI server **330** and sent to client **310** via a registered handler.

Event Handling by Server Application Program

FIGS. *5a–5g* show a process flow diagram of a typical event handler which may be registered by server application program **330** and service events generated by a background process for controlling the user interface using the messaging protocol of one embodiment of the present invention. For example, such an event handler may be registered using the Apple Event Manager described in *Inside Macintosh, Volume VI*, Chapter 6, wherein all of the above-described events are handled by this single handler **500**. Handler **500** will typically have a process entry point, such as **501** illustrated in FIG. *5a*, and comprise an IF or CASE programming statement in a typical high level programming language or other condition checking loop, which is illustrated in the remainder of the figures. Then, each of the events may be checked for, and upon detection of a specific event, the user interface display specified by the event and associated parameters may be displayed upon system display **121**. For example, it will be determined at step **502** whether a NewCopyWindow event has been detected. If so, then a new copy window (e.g., **400**) is displayed at step **503** with the specified parameters, such as statusString **410**, countString **420**, actionString **430**, objectName **440** , and having the progress bar **450** with an appropriate representation as defined by the progressValue/progressMax parameters passed within the event. Upon display of the new copy window **400** at step **503**, process **500** continues and returns at step **517**.

If, however, a NewCopyWindow event was not detected at step **502**, and a window is not currently displayed as

detected at step **504**, then an error condition may be indicated at step **505**, such as by issuing an event from server **330** to client **310** to specify an error (e.g., "EventNotHandled") to specify that the event was not serviced. Then, event handler **500** may exit at step **517**. If, however, a window has already been displayed, then the condition for the various remaining events defined in the messaging protocol may be checked for using a suitable programming construct such as a CASE statement or other similar condition-checking statement(s). For example, it is determined at step **506** whether the DisposeCopyWindow event has been detected. If so, then the copy window displayed upon computer system display **121** is eliminated at step **507** using well-known prior art user interface operations, and handler **500** returns at step **517**.

Upon the detection of a ChangeString event, as detected at step **508**, then process **500** proceeds to a more detailed sequence of steps to determine the value passed within the string number, as reflected on Figure *5b*. As is illustrated in FIG. *5b*, it is determined using a condition checking loop, such as a CASE statement or other programming construct, what the value of stringNumber is. For example, at step **520**, it is determined whether stringNumber indicates that the statusString should be modified. If so, then statusString **410** on display **400** is updated at step **521**, and the handler returns at step **517**. If, however, the countString is specified at step **522** (when stringNumber=1), then the count string (e.g., **420**) is updated at step **523**, and the handler returns at step **517**. If, however, the actionString should be modified (stringNumber=2), as detected at step **524**, then it is updated on the copy window. If, however, stringNumber=3 indicating that objectNameString **440** is sought to be updated, as detected at step **526**, then objectNameString **440** is updated at step **527**, and handler **500** returns at step **517** of FIG *5a*. Any other string number results in an error being generated at step **528**, and a return from the handler at step **517** with an appropriate error event message to client **310**, such as "EventNotHandled," indicating that the handler did not service the event.

Process **500** of FIG. *5a* proceeds to step **509** if a ChangeString event was not detected. Step **509** determines whether the ChangeBar event has been detected. If so, then handler **500** proceeds to step **510** which updates progress bar **450** using the progressvalue and progressMax parameters passed in the event. If the value(s) passed are invalid, an error may be indicated via a response event and the update to the progress bar abort.

If, however, the ChangeBar event is not detected at step **509**, then the handler proceeds to determine whether a PresentAlert event has been detected at step **511**. Various types of alerts can then be checked for, as illustrated in FIG *5c*, by checking the alertType parameter. For example, each of the steps illustrated at steps **530**–**540** on FIG. *5c* may be conformed using a typical high-level programming construct such as a CASE statement. Then, upon detection of the corresponding value in the alertType parameter, the associated alert is displayed. For instance, for alertType=0, as detected at step **530**, the generic alert is tested for and then displayed at step **531**. At step **532**, the NoteAlert is tested for (with alertType=1) and displayed at step **533**. At step **534**, the Confirm/Cancel alert is tested for (with the alertType=2), and it is displayed if detected at step **535**. At step **536**, if the Continue/Cancel alert type is detected (alertType=3), then the Confirm/Cancel option window is displayed at step **537**. At step **538**, the CancelDef aultConfirm option is tested for (alertType=4), and the corresponding option is displayed at step **539**. Finally, the SaveChanges parameter is tested for (alertType=5) at step **540** and then displayed at step **541**. If any other alertType value is detected, an error is indicated at step **543**, and the process returns at step **517** of FIG. *5a*. Otherwise, upon completion of detection of any of the above alertType values, then the option previously displayed is saved for use when the next event is detected in the event handler at step **542**, and process **500** returns at step **517** of FIG. *5b*.

At any rate, upon detection of the all the previous events, if the events are serviced, then an event message from server **330** to client **310** such as EventHandled is issued, and handler **500** returns at step **517**. Otherwise, any events detected which do not fall into one of the categories tested for or other events which are not serviced may issue a suitable error event message, such as EventNotHandled at step **517** to indicate to client **310** that the event was not serviced. Then, the client may take appropriate actions via its own event handler.

Thus, an invention for a background application controlling the user interface of a foreground application has been described. Although the present invention has been described particularly with reference to specific embodiments as illustrated in FIGS. 1–5*c*, it may be appreciated by one skilled in the art that many departures and modifications may be made by one of ordinary skill in the art without departing from the general spirit and scope of the present invention.

What is claimed is:

1. In a computer system comprising a processor, a display, a memory, a user input device, a first process operative in the computer system, a second process operative in the computer system as a foreground process and a user interface on said computer system display under the control of the second process, a method for the first process to perform operations for the second process and control a content of the user interface on said computer system display, said content under control of the foreground second process operative in said computer system, said first process controlling the content to display information regarding the operations performed by the first process for the second process, said method comprising the following steps:

a. installing an event handling process as part of said second process, said event handling process when said second process is operative in said computer system, servicing events generated by the first process for controlling said user interface display under control of said second process;

b. said second process initiating said first process to perform operations for said second process, said second process operative in the foreground and said first process operative in the background;

d. said first process generating events for controlling said user interface display while the second process remains as a foreground process and the first process is a background process, said events providing information regarding the operations performed by said first process for the second process; and

e. said event handling process receiving events generated by said first process, said event handling process updating said user interface on said computer system display according to said events generated by said first process, while said first process remains in the background, and received by said event handling process.

* * * * *

EXHIBIT G

US006343263B1

(12) **United States Patent**
Nichols et al.

(10) **Patent No.: US 6,343,263 B1**
(45) **Date of Patent: Jan. 29, 2002**

(54) **REAL-TIME SIGNAL PROCESSING SYSTEM FOR SERIALLY TRANSMITTED DATA**

(75) Inventors: **James B. Nichols**, San Mateo; **John Lynch**, San Jose, both of CA (US)

(73) Assignee: **Apple Computer, Inc.**, Cupertino, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **08/284,061**

(22) Filed: **Aug. 2, 1994**

(51) **Int. Cl.**[7] **G06F 13/00**
(52) **U.S. Cl.** **702/189**; 709/328
(58) **Field of Search** 364/514, 238.5, 364/924, 576, 725, 726; 395/2.09, 2.1, 2.91, 2.94, 406, 651, 680, 682, 821, 892, 561, 566, 733; 455/84, 3.1, 39; 370/210; 340/825; 342/195, 196; 360/39

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,327,558	A	*	7/1994	Burke et al.	395/650
5,363,315	A	*	11/1994	Weiss et al.	364/514
5,381,346	A	*	1/1995	Monaham-Mitchell et al.	364/514
5,406,643	A	*	4/1995	Burke et al.	395/200
5,438,614	A	*	8/1995	Rozman et al.	379/93
5,440,619	A	*	8/1995	Cann	379/97
5,440,740	A	*	8/1995	Chen et al.	395/650
5,442,789	A	*	8/1995	Baker et al.	395/650
5,487,167	A	*	1/1996	Dinallo et al.	395/650

FOREIGN PATENT DOCUMENTS

EP	218859	4/1987

OTHER PUBLICATIONS

Tanenbaum, *Structured Computer Organization*, 1984, pp. 10–12.*

Frankel, "DSP Resource Manager Interface & Its Role in DSP Multimedia", May 1994.*

Silberschatz et al., *Operating System Concepts*, p. 489, 1994.*

Jon Udell, "Computer Telephony," *Byte*, vol. 19, No. 7, Jul. 1994, pp. 80–96.

* cited by examiner

Primary Examiner—Patrick Assouad
(74) *Attorney, Agent, or Firm*—Burns, Doane, Swecker & Mathis, L.L.P.

(57) **ABSTRACT**

A data transmission system having a real-time data engine for processing isochronous streams of data includes an interface device that provides a physical and logical connection of a computer to any one or more of a variety of different types of data networks. Data received at this device is presented to a serial driver, which disassembles different streams of data for presentation to appropriate data managers. A device handler associated with the interface device sets up data flow paths, and also presents data and commands from the data managers to a real-time data processing engine. Flexibility to handle any type of data, such as voice, facsimile, video and the like, that is transmitted over any type of communication network with any type of real-time engine is made possible by abstracting the functions of each of the elements of the system from one another. This abstraction is provided through suitable interfaces that isolate the transmission medium, the data manager and the real-time engine from one another.

41 Claims, 6 Drawing Sheets

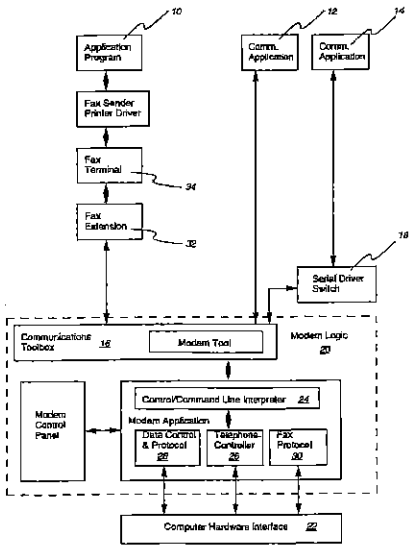


FIGURE 1

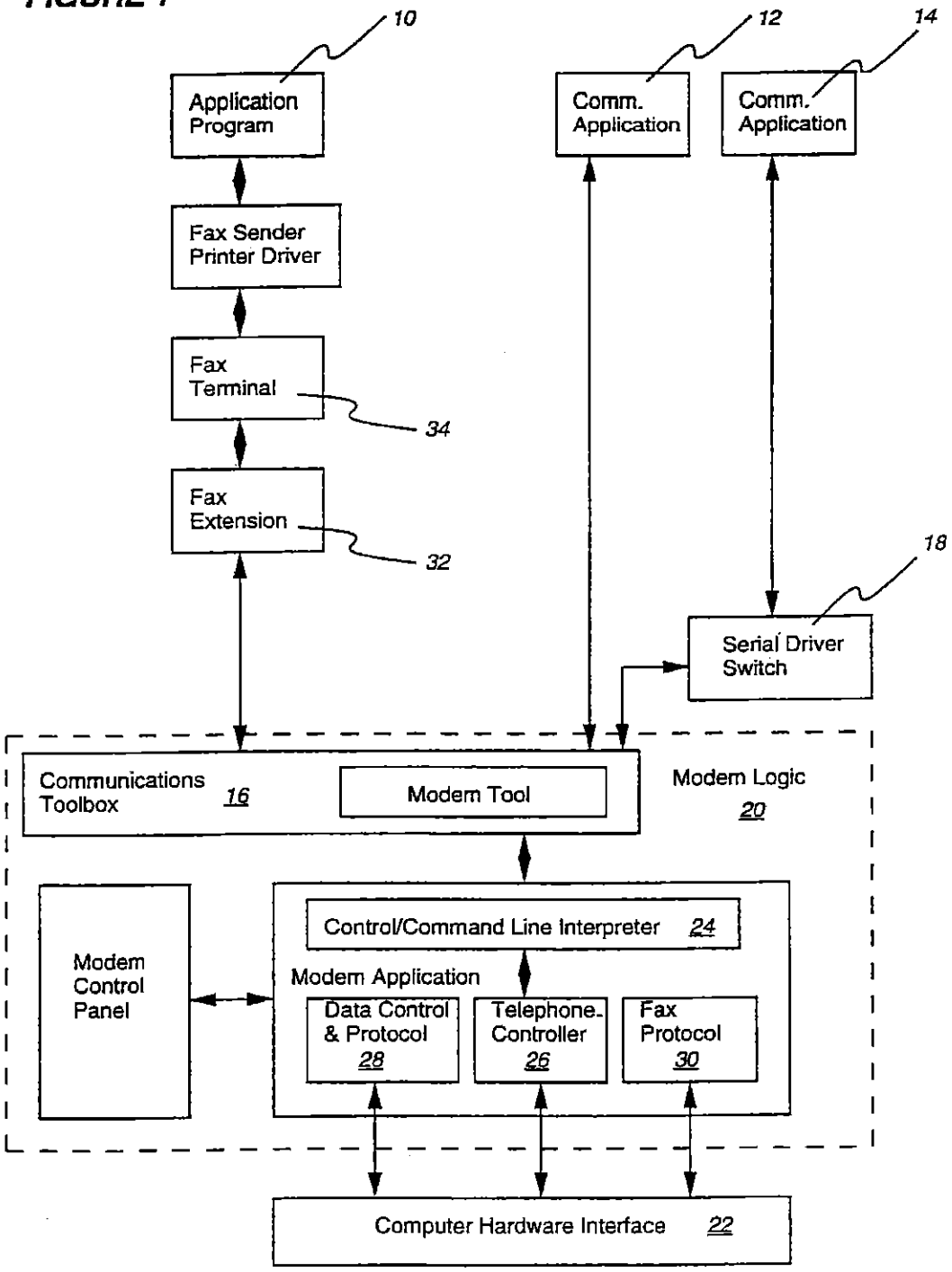

10
12
14
Application Program
Comm. Application
Comm. Application
Fax Sender Printer Driver
Fax Terminal
34
Fax Extension
32
18
Serial Driver Switch
Communications Toolbox 16
Modem Tool
Modem Logic 20
Modem Control Panel
Control/Command Line Interpreter 24
Modem Application
Data Control & Protocol 28
Telephone Controller 26
Fax Protocol 30
Computer Hardware Interface 22

FIGURE 2

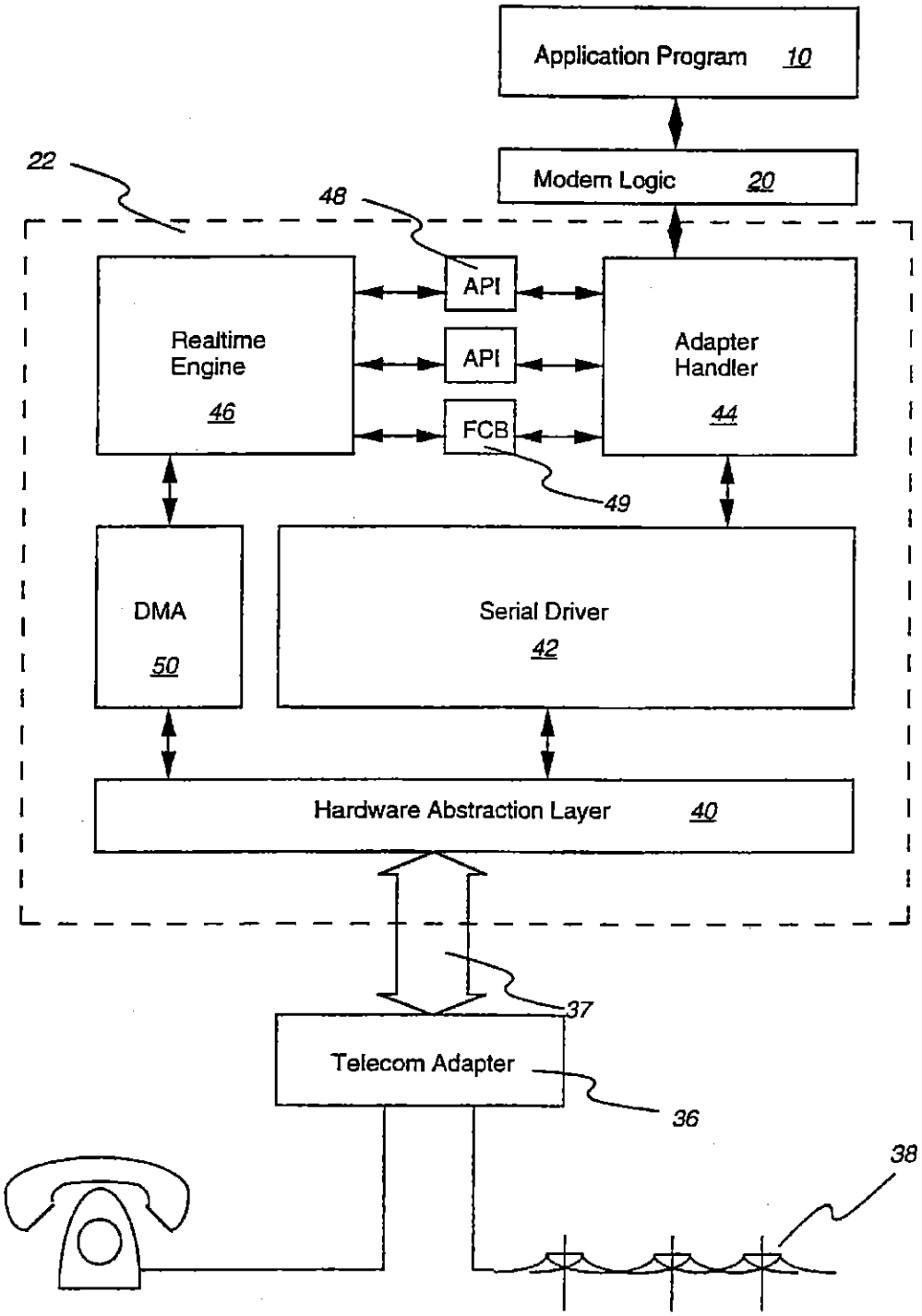
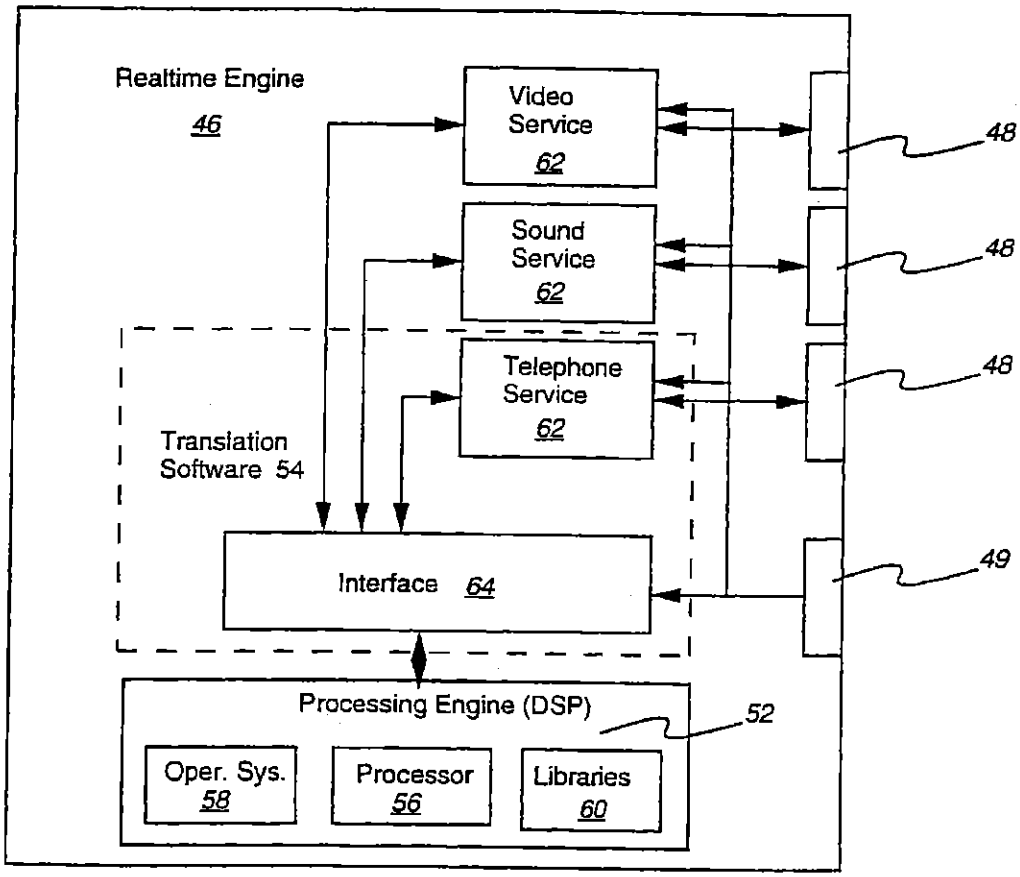

Application Program 10
Modem Logic 20
22
48
API
API
FCB
Realtime Engine 46
Adapter Handler 44
49
DMA 50
Serial Driver 42
Hardware Abstraction Layer 40
37
Telecom Adapter
36
38

FIGURE 3


Realtime Engine
46
Video Service
62
Sound Service
62
Telephone Service
62
Translation Software 54
Interface 64
Processing Engine (DSP)
52
Oper. Sys.
58
Processor
56
Libraries
60
48
48
48
49

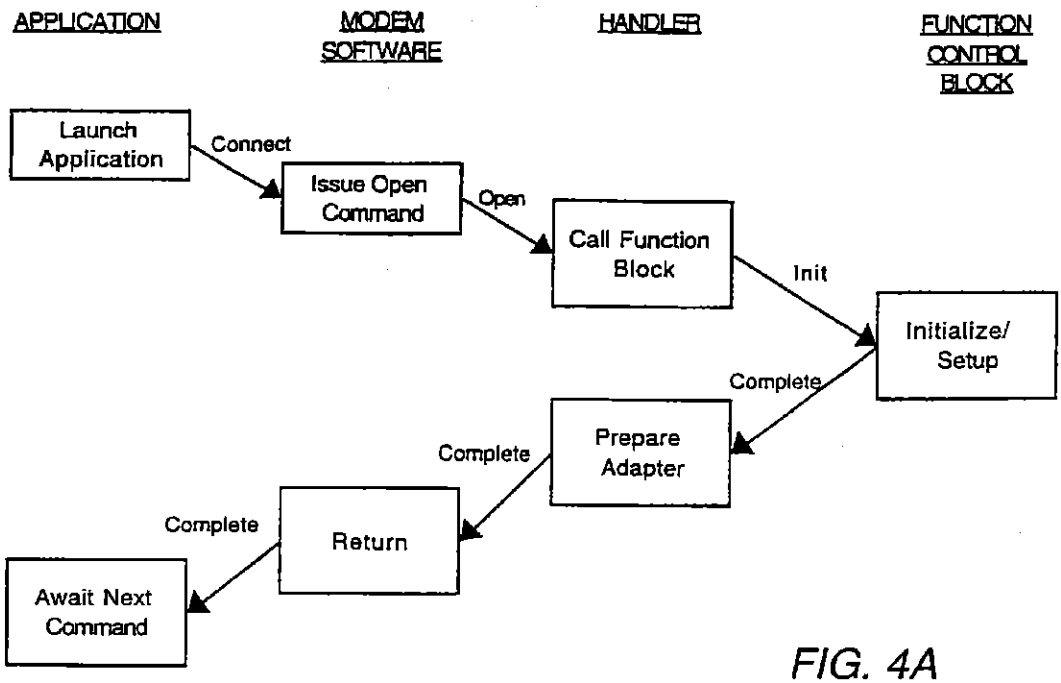

APPLICATION
MODEM SOFTWARE
HANDLER
FUNCTION CONTROL BLOCK
Launch Application
Connect
Issue Open Command
Open
Call Function Block
Init
Initialize/ Setup
Complete
Prepare Adapter
Complete
Return
Complete
Await Next Command

FIG. 4A

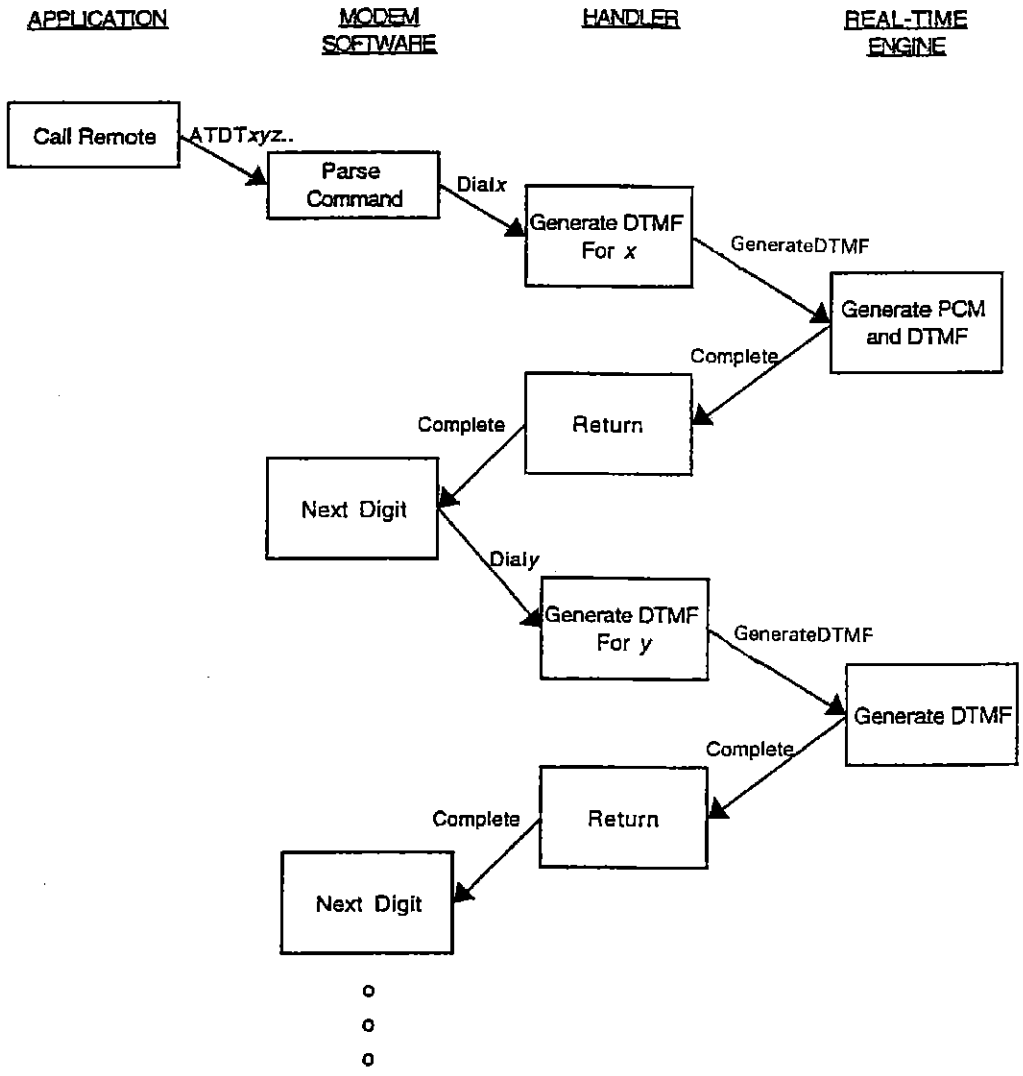

APPLICATION
MODEM SOFTWARE
HANDLER
REAL-TIME ENGINE
Call Remote
ATDTxyz..
Parse Command
Dialx
Generate DTMF For x
GenerateDTMF
Generate PCM and DTMF
Complete
Return
Complete
Next Digit
Dialy
Generate DTMF For y
GenerateDTMF
Generate DTMF
Complete
Return
Complete
Next Digit
o
o
o

FIG. 4B

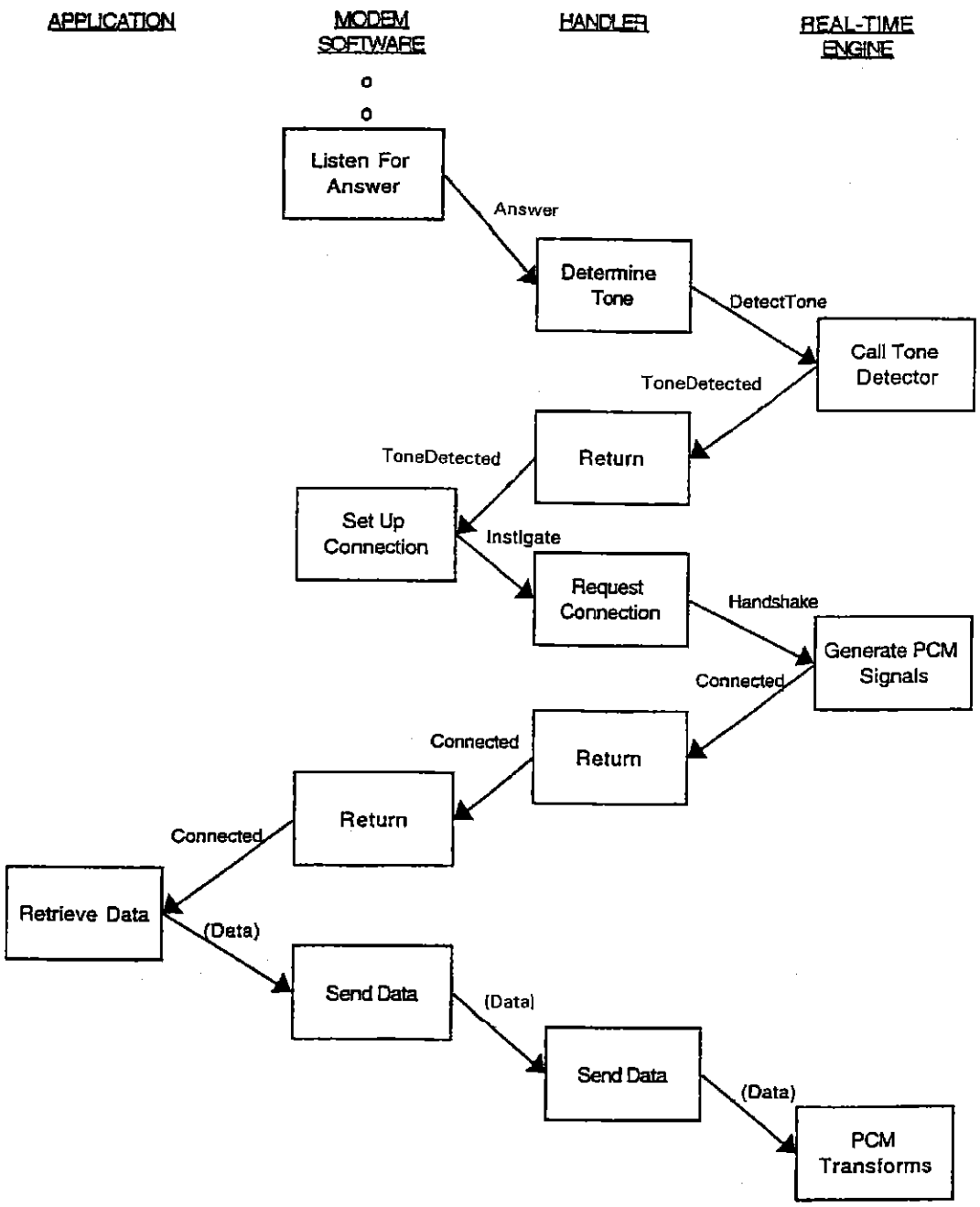

APPLICATION
MODEM SOFTWARE
HANDLER
REAL-TIME ENGINE
o
o
Listen For Answer
Answer
Determine Tone
DetectTone
Call Tone Detector
ToneDetected
Return
ToneDetected
Set Up Connection
Instigate
Request Connection
Handshake
Generate PCM Signals
Connected
Return
Connected
Return
Connected
Retrieve Data
(Data)
Send Data
(Data)
Send Data
(Data)
PCM Transforms

FIG. 4C

REAL-TIME SIGNAL PROCESSING SYSTEM FOR SERIALLY TRANSMITTED DATA

FIELD OF THE INVENTION

The present invention is directed to the transmission of data to and from a computer, and more particularly to a system for performing real-time signal processing of data that is serially transmitted to and from a computer.

BACKGROUND OF THE INVENTION

Various devices are known for transmitting data between a computer and a remote site via wide-area telecommunications networks. One of the most widely used devices of this type is the modem, which enables data to be transmitted to and from a computer over a wide-area analog telephone network. Generally speaking, the modem includes one or more sets of registers, typically embodied in an UART or an USART, for storing bits of digital data transmitted to or from the computer, a processor for implementing modem operations, such as dialing a telephone number or answering a ringing signal, in response to commands sent from the computer and stored in the UART, and a modulator/demodulator for converting digital bits of data to be transmitted into analog signals, and vice versa. Originally, all of these features were hardwired in a separate peripheral device that could be externally connected to the computer via a serial I/O port, or internally connected to the computer's data bus. More recently, some of the functions associated with these features, most notably the processing of commands to implement modem operations, have been removed from the hardwired configuration and incorporated into the computer itself. This approach has increased the versatility of the modem. For example, while the hardwired modem configuration had to be specifically designed for the telephone system requirements of a particular country, the later approach could enable a single product to be used in a variety of countries, each of which might have different telephone signaling requirements. Similarly, since the computer itself was handling the data to be transmitted, additional services, such as the ability to send information as a facsimile transmission, in which graphical data is processed, became feasible. However, the presence of the UART, or similar such device through which the data must flow, still limits the effective rate at which the data can be exchanged between the computer and the telephone lines.

To enhance the performance of modems, a digital signal processor (DSP) has been incorporated into its structure. In this arrangement, the modem software was designed to cooperate with the DSP to provide data thereto for processing prior to transmission or after reception over the telephone line. While the addition of the DSP provided increased capabilities in terms of the speed at which the data could be transmitted over a telephone network and the ease with which the modem could be configured, it was still limited in the types of data that could be processed. More particularly, because of the restrictions imposed by passing the data through an UART or the like, even the most modern modems are only capable of effectively transmitting data in the range of 9.6–14.4 Kb/sec. While this rate of data transfer may be useful for transmitting static information such as text files or the like, it is not suitable for many real-time applications in which the data is provided at much higher rates, such as speech or video conferencing.

Further in this regard, the modem control software had to be designed to work with the specific DSP incorporated into the computer. If a different DSP was to be used, the modem control software had to be reprogrammed to work with the new DSP.

While the analog telephone network was the only practical medium for transmitting information between geographically distributed computers for quite some time, more recently other, non-analog transmission mediums have become available. Examples of these other mediums include the integrated services digital network (ISDN), private branch exchange (PBX) telephone systems, and TI digital data links. Since information is transmitted over these mediums as digital data, conventional analog modem circuits are not suited for use with them. Thus, for example, a standard Group III facsimile machine cannot operate on a digital PBX system.

Similarly, digital signal processing systems which are designed to work with PCM-encoded analog data that is received and transmitted via a modem are likewise not suited for use with these other types of transmission mediums. While it is possible to incorporate another DSP system into a computer that can handle data transmitted via any of these digital networks, it would be more desirable to provide a single system that can process data that is received over any type of transmission medium, whether it be digital or analog. Further in this regard, it is desirable to provide a signal processing system that is not limited to one specific DSP, but rather one that can operate with any of a variety of different types of signal processors.

When personal computers communicate with one another through non-modem transport facilities, they typically operate in a burst mode. While this mode of operation enables data to be transferred at much higher rates than with modems, it is still not suitable for applications such as video or speech processing. These types of applications require isochronous data handling, i.e. data that is transmitted at a constant bit rate and that must be processed in real time. Generally speaking, therefore, it is desirable to provide a serial data transmitting and receiving system that is capable of processing real-time isochronous data.

Further in this regard, it is desirable to provide such a system that is capable of handling streams of data pertaining to different functions. For example, in a video conferencing application, speech data is transmitted at the same time as video and other graphic information. However, each of these types of data must be processed separately in real time. It is desirable, therefore, to provide a data transfer system that can handle each of two or more types of data at isochronous rates.

BRIEF STATEMENT OF THE INVENTION

The present invention provides a data transmission system having a real-time data engine for processing isochronous streams of data. An interface device provides a physical and logical connection of the computer to any one or more of a variety of different types of data networks, including analog telephone, ISDN, PBX and the like. Data received at this device is presented to a serial driver, which disassembles different streams of data for presentation to appropriate data managers, such as the operating system of the host computer, a service provider or an application program. A device handler associated with the interface device sets up data paths and issues service requests. The device handler also presents data and commands from the data managers to a real-time data processing engine, that can be used for a variety of applications such as voice recognition, speech compression, and fax/data modems. This real-time engine can be shared by any application program running on the host computer.

The invention enables any arbitrary type of data, such as voice, facsimile, multimedia and the like, which is trans-

mitted over any type of communication network, to be handled with any type of real-time engine, by abstracting the functions of each of the elements of the system from one another. This abstraction is provided through suitable interfaces that isolate the transmission medium, the data managers and the real-time engine from one another. The data is provided to the real-time engine as multiple streams of isochronous data, i.e. it is delivered as it arrives over the network without data handling delays. This feature allows more complex applications, such as speakerphones, videophones and high-speed modems to be readily implemented.

These features of the invention, as well as the advantages offered thereby, are described in greater detail hereinafter with reference to a specific embodiment illustrated in the accompanying figures.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram of the software architecture for a facsimile/modem communications engine;

FIG. **2** is a block diagram of the computer hardware interface for the engine;

FIG. **3** is a more detailed block diagram of the architecture of the real-time engine; and

FIGS. **4A–4C** are flow diagrams of the steps that are carried out in the operation of a virtual fax/data modem in accordance with the invention.

DETAILED DESCRIPTION

The following description of an embodiment of the present invention is presented in the context of its implementation in a moderate speed (e.g. T1 data rates), low cost, digital interconnect to a wide-area network. An example of such an interconnect is described in U.S. patent application Ser. No. 08/180,926 filed Jan. 11, 1994, now U.S. Pat. No. 5,515,373 the disclosure of which is incorporated herein by reference. The practical applications of the invention are not limited to this particular embodiment, however. For example, it can also be employed as a high-performance interconnect to multimedia devices such as digital imaging equipment and audio appliances.

To facilitate an understanding of the invention, it will be described with reference to the specific example of a telephone-based telecommunication subsystem that provides basic fax/data modem services, plus call management and audio stream handling. This particular example is perhaps one of the more complex applications of the invention, because of the number of different signaling requirements associated with communications over the telephone lines. Further complexity is added by the fact that these requirements are often country-specific, and therefore the handling of a command, such as dialing a telephone number, can vary greatly from one country to the next. Other implementations of the invention, for example in the context of transmitting sounds and video data, will become apparent from an understanding of the principles of the invention explained with respect to this particular example.

FIG. **1** is a block diagram of a communications engine that is designed to provide modem and facsimile services. The top level of the diagram represents communications applications **10–14**. These applications send commands for communications services to a data service provider, or application programming interface (API), such as the Telephone Manager provided by Apple Computer, Inc. or the Telephone Application Programming Interface (TAPI) provided by Microsoft Corp. This interface is represented by the communications toolbox **16** contained within fax/data modem logic modules **20**. A hardware interface **22** transmits the services provided by these modules over a transmission medium to which the computer is connected, e.g. a telephone line.

Referring to the modem logic **20**, a Modem Control and Command Line Interpreter (CLI) **24** accepts command inputs from the application programs to configure the modem, dial, receive calls, initiate data or fax transmission, hang up, and so on. Any operation of the modem must be initiated by issuing commands to the CLI. The CLI **24** can be an interface that is often referred to as the "AT command set", which uses simple printable character sequences and constitutes a de facto standard among modems.

A Telephone Controller module **26** dials, answers, and hangs up the telephone line. The Telephone Controller can be "worldwide" in nature. In such a case it configures the modem to conform to the standards of telephone systems in most major economic markets. This can be carried out by storing a country code identifier that allows the Telephone Controller to configure the modem properly.

A Data Control and Protocol module **28** provides all data capabilities for the modem. It supports standard asynchronous text read and write, as well as the standard CCITT V.42 and V.42 bis and Microcom MNP class 2 through 5 reliable link and data compression protocols, for example.

The Fax Protocol module **30** performs the functions of a T.30 fax engine in a dedicated fax modem. It communicates with a Fax Extension driver **32**, a component of a Fax Terminal **34**, for example using an extension of the AT command set known as +F ("plus F"), or TR.**29**.

The hardware interface **22** contains the appropriate transport-dependent protocol components. The logical and physical interface to the wide-area network is hidden in this layer. This allows the fax/data modem modules to be used on any wide-area connection, including PBX, T1 and ISDN, as well as traditional POTS ("plain old telephone system") channels. The components of this interface are illustrated in the block diagram of FIG. **2**.

Referring now to FIG. **2**, the hardware interface includes an external adapter **36** that provides the physical and logical connection of the computer to the telephone line **38** or other communications network, such as ISDN, PBX or T1 link. This connection can be provided through a serial port **37** of the computer. An example of such an adapter is described in copending U.S. patent application Ser. No. 08/078,890 filed May 10, 1993, and Ser. No. 08/180,925 filed Jan. 11, 1994, the disclosures of which are incorporated herein by reference. Such an adapter preferably includes processing capabilities, as disclosed in concurrently filed U.S. patent application Ser. No. 08/288,144 now U.S. Pat. No 5,799,190 entitled Intelligent Communications Coprocessor, which enables it to provide a constant stream of data to the computer from one or more communications networks. This data can be delivered in a time-division multiplexed manner, or it can be delivered in a packetized form.

Data received at the adapter **36** from the telephone line **38** can be provided to a driver **40** which functions as a hardware abstraction layer. This driver is a hardware-dependent implementation of a serial controller, and is configured in accordance with the particular characteristics of the communications port **37** to which the adapter **36** is connected. It isolates the remainder of the software from the different implementations for connecting the adapter to the computer, e.g. serial port, parallel port or bus device.

A serial driver **42** operates to separate multiple incoming data streams from one another, or to combine multiple

outgoing data streams that are intended for respective transmission to separate destinations, whether logical or physical. For example, the adapter **36** might be connected to a desktop telephone for voice communications and to the telephone line for wide-area communications. Data streams for these two connections can be combined in a multiplexed manner by the serial driver, to be sent to the adapter **36** through the computer's serial port **37**.

An adapter handler **44** is specific to the particular adapter **36** and carries out features associated with that adapter. For example, if the adapter is one that is designed to be connected to a telephone network, the handler implements functions germane to that network such as ring detection, pulse dialing, on/off hook control and the like, in response to commands received from the application programs **10**. One example of the manner in which data can be exchanged between the adapter **36** and the handler **44** is described in U.S. patent application Ser. No. 08/058,750 filed May 7, 1993, now abandoned the disclosure of which is incorporated herein by reference.

A real-time engine **46** can perform transforms on data streams provided to and received from the adapter **36**. The particular transforms to be performed are sent as commands to the real-time engine from the adapter handler **44** via suitable application programming interfaces **48**. For communicating with the real-time engine, each interface includes shared command/control mailboxes in the computer's RAM, as well as bi-directional first-in, first-out (FIFO) buffers for transferring data. As an example, if the system is set up to operate as a fax/data modem, the real-time engine functions as a virtual telephone. In such a case, the handler may instruct the engine **46** to send a facsimile in response to a command from an application program. For this purpose, the real-time engine is configured with a suite of modem and call processing functions. This configuration is implemented by a real-time function control block **49**, which initializes and manages the operation of the real-time engine. Generally speaking, whenever a new task is to be carried out by the real-time engine **46**, the real-time function control block **49** issues commands that are specific to the operating system of the real-time engine. These commands cause the engine to start up, if it is not already running, and to configure itself with a library of routines that are necessary for it to implement the task.

When the handler **44** requests a facsimile transmission, for example, the real-time function block issues commands to start the real-time engine and install the various modules that are needed for it to function as a virtual telephone. Binary facsimile image data is transferred to the real-time engine via the FIFO buffers, where it is encoded as PCM data which is further encoded according to the transport medium over which it is to be transmitted. If the adapter is connected to a telephone line, for example, these signals can be encoded as 16-bit pulse-code modulated (PCM) samples, and forwarded directly to the adapter **36** via the serial driver **42**. Alternatively, if the transport medium is an ISDN line, the modem signals are encoded as mulaw-companded 8-bit PCM signals. The different types of encoding are stored in different tables, and the appropriate one to be used by the real-time engine is installed by the real-time function block during the initial configuration of the engine and/or designated by the API **48** at the time the command to transform the data is issued.

Transformed signals from the real-time engine that are to be transmitted via the transmission medium are provided to the hardware abstraction layer **40** through direct memory access (DMA) **50**. When data is being received from the communications network, it is provided to the real-time engine as pulse-code modulated (PCM) data through the DMA **50**. If the computer does not have DMA capabilities, the data can be transferred between the adapter and the handler as packetized data, as described in application Ser. No. 08/058,750 filed May 7, 1993 now abandoned.

As another example, the real-time engine may operate as a virtual speech recognition device. To do so, the real-time function block initializes the engine and installs the modules necessary to carry out this function. In this mode of operation, the engine may convert pulse-code modulated (PCM) data received from the adapter into vector-quantized speech components. The engine transforms this PCM data into data appropriate for subsequent processing by a speech recognition application program, according to the commands from the handler **44**, and the results of the transforms are provided through the shared FIFOs.

With this configuration, the handler does not need to have any knowledge of the real-time engine implementation. Communications with the real-time engine are carried out in a robust fashion. In essence, the architecture of the system provides a message-passing capability between devices that know nothing about the configuration or existence of one another.

As illustrated in FIG. **2**, there can be a number of interfaces **48** situated between the handler **44** and the real-time engine **46**. Each interface represents services for a particular class of functionality. For example, one interface may relate to the operation of the engine as a virtual telephone, another interface can be associated with a virtual sound device, e.g. stereo, and a third interface can pertain to a virtual video device. Each interface receives commands from an application program, through the handler **44**, and instructs the real-time engine to carry out the necessary transforms which relate to the function of the virtual device being implemented, e.g. text-to-speech conversion, video image processing, etc.

The architecture of the real-time engine **46** is illustrated in further detail in FIG. **3**. Referring thereto, when configured as a virtual device, the real-time engine is made up of two main components, a processing engine **52**, such as a DSP, and translation software **54**. The DSP comprises a processor **56**, an operating system **58** for that processor, and a set of libraries **60** which enable the processor to perform designated signal processing functions. There are three possible implementations of the DSP, respectively identified as hard, soft and native. In the hard implementation, all three components of the DSP are fixed within a piece of hardware, i.e. an IC chip. In other words, the libraries and the operating system are embedded as firmware, and cannot be reprogrammed or updated without changing the chip. An example of a hard implementation is the Rockwell 9623 data-pump. This type of DSP might be able to perform only one class of virtual device operation, i.e. Function as a modem. When a hard DSP is employed for the real-time engine, the function control block **49** operates to initialize the processor at the outset of the operation of the real-time engine.

The soft implementation differs from the hard implementation in that the libraries, and possibly also the operating system, are resident as software in the computer's memory. In this implementation, the libraries are programmable and can be updated as desired. The processor, however, is still resident as a separate piece of hardware. Because of this programmability, the DSP can carry out more functions than a hard DSP, such as sound processing and Fourier transforms. An example of a soft DSP is the AT&T 3210. When

a soft DSP is employed for the real-time engine, the function control block **49** operates to configure the appropriate libraries for the transforms that are to be carried out by the DSP, in addition to initializing the processor.

In the native implementation of the DSP, the processor does not reside in a separate piece of hardware. Rather, it's functions are carried out by the CPU of the host computer. As in the case of the soft implementation, the DSP operating system and the libraries are resident in the computer's memory. When this implementation is employed, the function control block **49** operates to allocate system resources to the DSP function, such as to enforce system time management, to ensure that adequate processing time is given to DSP operations.

The translation software **54** is made up of two parts. The main part of the software comprises a generic service provider **62** which functions as a device driver. This part of the software receives the commands from one of the APIs **48** and issues the instructions to the DSP to perform the transforms that are required in the operation of the virtual device being implemented. This part of the software is labeled as being generic because it is independent of the actual hardware that is used in the implementation of the DSP **52**. To enable the service provider to communicate with the DSP, an interface **64** is provided. This interface is specific to the particular DSP that is employed as the processing engine for the real-time engine. In other words, the generic service provider does not need to know whether the processing engine is a hard, soft or native DSP. In essence, therefore, the interface **64** functions as an additional layer of abstraction which virtualizes the DSP, i.e. the generic service provider is aware of the existence of a DSP, but does not need to know how it is actually being implemented in order to operate.

In a practical embodiment of the invention, separate generic service providers can be employed for the different virtual devices to be implemented. For example, one service provider can be employed to provide the services of a virtual telephone. Such a service provider might include a set of calls which enable it to determine the capabilities of the hardware being employed, e.g. whether it can support line current detection, remote wakeup, etc. Other sets of calls are used for control and status information, tone generation and detection, data transfer, and power management.

Another service provider can be used for sound applications, and a third service provider for video applications. Depending upon the particular virtual device to be implemented, the function control block **49** calls up the appropriate service provider when configuring the real-time engine. Each service provider communicates with the handler **44** through a respective one of the APIs **48**, and with the DSP **52** through the same interface **64**.

An example of a suitable interface **48** for telephony applications will now be described in detail. The interface basically operates to transmit high-level requests for service. The functionality of such an interface can be divided into two main categories, namely functions that are used only on public service telephone network (PSTN) lines, such as ring detection, and those functions used on any telephone line, such as DTMF generation and detection. This arrangement allows the interface to be used for ISDN and PBX lines as well as traditional analog lines for call progress and modem functions. For PSTN lines, the interface generates commands for setting the appropriate electrical parameters, such as voltage levels that comply with a particular country's regulations. For this purpose, the interface can include data tables containing information on all country-specific parameters.

The interface provides for the origination and answering of calls routed through traditional analog switches. To answer calls, the interface monitors incoming signals, as reported by the real-time engine, for appropriate frequency and cadence consistent with a particular country's requirements. The interface also includes facilities for tracking call progress, such as detection of dial tone, ring back and busy signals. It further includes the necessary information relating to the generation and detection of DTMF signals.

The interface generates calls that can be classified into two general categories, originating calls and callback calls. Originating calls are those which are generated in response to commands from the application program. Callback calls are used to report progress information to the application program. Most originating calls might take time to complete, and are therefore asynchronous, so that the host processor can suspend servicing of the calling application program until the task associated with the call is complete. This allows the interface to be called from within an interrupt handler as well as freeing the processor while waiting for some hardware to execute a task. Completion of the process is indicated by executing a completion routine for an associated callback.

The originating calls are of two types, system task calls and general purpose calls. The system task calls can include those such as "Open", which causes system resources to be allocated to the real-time engine, and "Close", which deallocates the resources. General purpose calls can include such calls as "State", which returns the current state of the virtual device, e.g. on-hook, ringing, off-hook or on-line, "GenerateDtmf" which causes DTMF tones to be generated, and "SetAutoAnswer", which instructs the engine to answer a call after a predetermined number of rings. Other examples of general purpose calls include "SetSilenceDuration", which passes to the engine the length of a silence to be detected, "Hook", which is used to take the virtual phone off-hook and on-hook, and "DialNumber", which dials a number in a designated string. Similar types of general purpose calls can be included for functions associated with facsimile types of operations.

Examples of suitable callback calls include "DtmfDetected", which indicates that a particular DTMF digit has been detected, "RingIndicate", which identifies when a valid ring has been detected on the line, and "DialToneDetected", which is called when a valid dial tone is detected. Other appropriate callback calls will be readily apparent from these examples.

The flow of events that occur when the fax/data modem is activated will now be described. These events are illustrated in the flow diagrams of FIGS. **4A–4C**. From the perspective of an application program **10**, it is "talking" to an external modem connected to the serial port. In fact, however, it is actually obtaining communications services from an internal virtual modem **20** and the hardware interface **22**.

At boot time, the computer's operating system determines whether the computer is capable of supporting a communications system. This allows the operating system to notify the user on application program activation that a communications session is not possible on the computer. This determination is made by assessing system-dependent factors such as presence of a data stream processor, sufficient system resources, and so forth. In the following discussion it is assumed that the requisite resources are available.

At the outset, with reference to FIG. **4A**, a communications application **10** is launched. A communications "con-

nection" is opened either implicitly on launch, or by command. This directs the communications subsystem to initialize itself. The communications application's connection establishment request is passed to the communications toolbox **16**. This in turn causes a driver command "Open" to be issued by the fax/data modem logic **20** to the hardware interface **22**.

The hardware interface driver command "Open" is received by the hardware interface adapter handler **44**. As noted, the handler has previously—typically at boot time—determined that the host computer has the resources to establish a communications session, in this example being an analog modem over a telephone service line.

The handler calls function control block **49**, to initialize the real-time engine. The action taken by function control block depends on the real-time engine's implementation. If a programmable DSP is used for the real-time engine, the function control block might issue a series of DSP operating system specific commands to download and initialize the DSP subsystem, followed by commands to download the DSP algorithms that perform the modem's analog modulation. A native-mode DSP implementation may result in the function control block simply allocating system memory and host processor resources needed for the modem algorithms. In either case, it is significant to note that the hardware interface is designed to use a virtual real-time engine. The entire real-time engine implementation is "hidden" from the handler **44** by the function control block. The handler does not communicate with the real-time engine directly, via DSP-specific commands. Rather, all communications take place over the virtual real-time engine interface **48** via the mailboxes and the full-duplex data FIFO registers.

The handler **44**, as part of the Open process, prepares the attached telecom adapter **36** for operation. After initialization, the hardware interface telecom adapter delivers a full duplex, isochronous data stream.

If the real-time engine is successfully configured, the hardware interface is initialized properly, and all other necessary resources are available, the handler Open operation will be successful, and an analog modem communications link over the hardware interface adapter can begin at any time. Otherwise, communications are impossible and an error is reported to the application program.

Referring now to FIG. **4**B, the application program now initiates a modem connection with a remote station. For this purpose, the character sequence "ATDT5551212" might be issued by the application program, signifying (in the "AT" modem command standard) that the communications subsystem should dial the remote station at number 5551212, and instigate a modem connection with the answering modem, if present.

The dial command string is passed by the communications toolbox **16** to the CLI **24**, where it is parsed and converted into a virtual phone dial command. The virtual phone dial command is passed to the adapter handler **44** as a driver-level control call, where it is translated into a virtual real-time engine command and placed on the virtual real-time engine's command/response interface **48**. This causes the real-time engine to generate a PCM data stream that directs the hardware interface telecom adapter to go off-hook, then to generate the DTMF tones corresponding to the entered phone number. The handler **44** "sleeps", waiting for the real-time engine to signal that the real-time DTMF command has been completed. In this operation, the handler has no involvement with the isochronous data stream created by the real-time engine.

Referring now to FIG. **4**C, upon completion of the virtual phone dial command, the fax/data module issues a command to the virtual phone, i.e. the hardware interface, directing it to listen for an answering modem sequence. Again, this command is received by the handler **44** and is translated into a virtual real-time engine command to detect tones (signal energy) at certain specified frequencies and levels. Optionally, the real-time engine may be commanded to concurrently listen for voice energy in case a human answers the phone.

Assuming that the answering station presents valid answering modem tones, the handler will then be directed to instigate a modem connection. This results in another command to the virtual real-time engine, this time requesting a modem connection compatible with the answer tone sequence received. The actual modulation and demodulation of the hardware interface adapter's isochronous PCM data stream is accomplished entirely by the real-time engine.

Once the modem connection is established, the handler notifies the fax/data modem module that data transmission may begin. Digital data now flows between remote and local computers via handler Read and Write calls. Data is passed between the real-time engine and the handler via full-duplex FIFOs. This data is in turn passed between the handler and the application program through the modem logic **20**.

The ability to communicate over different types of transmission mediums in this single system is made possible by the fact that each of the various components is isolated from the particular features of the other through suitable levels of abstraction implemented via the application programming interfaces. For example, to change the transmission medium from the telephone lines to an ISDN line, the telecom adapter **36** is disconnected from the serial port **37**, and a new adapter appropriate for ISDN is plugged into the serial port. The associated adapter handler **44** is also loaded into the system. Thereafter, whenever the adapter handler issues a command to the real-time engine to perform a transform, it identifies the fact that the transformed data must be suitable for ISDN format. In response thereto, the API **48** which receives these commands supplies the real-time engine with the appropriate parameters for performing the transforms in the required format, e.g. the proper number of bits per word, etc.

Similarly, if the computer is transported from one country to another, the only change that needs to be implemented to carry out telephone communications in the new country is to switch the adapter and its handler. Upon initialization, the adapter identifies the fact that it is designed for a specific country. Whenever commands are to be sent to the real-time engine, the handler instructs the API **48** of the country as well as the command itself. For example, the command might be to generate a dial tone for country X. In response, the API **48** instructs the real-time engine to generate the dial tone, and provides it with the parameters pertinent to dial tones in country X. The real-time engine then generates the necessary PCM signals and supplies them to the adapter **36** via the DMA. The adapter takes care of converting those signals into the necessary electrical signals for transmission on the telephone lines of that country.

In essence, the real-time engine allows any type of transform to be performed on any type of data delivered over any type of transmission medium. The application program which receives the transformed data does not have to have any knowledge of the fact that the transmissions are being carried out over an ISDN line, rather than the telephone lines that it might have been originally programmed for. Thus, for

example, a modem connection can be established over an ISDN line without the application being aware of a change in the transmission medium. The adapter **36**, the hardware abstraction layer **40**, the serial driver **42** and the adapter handler **44** function to configure a real-time data stream from the transmission medium to the real-time engine, and vice versa. The speed at which this data can be delivered, as well as the format of the data, is no longer limited by hardware devices that are employed in conventional hard-wired modems, particularly UARTs and the like. Rather, the data is delivered at a real-time rate, where it is handled by the computer's CPU.

The foregoing examples of the invention have been presented to facilitate an understanding of its features and operation. It will be appreciated, however, that the practical applications of the invention are not limited to these specific embodiments. Rather, the invention will find utility in any environment in which it is desirable to transmit and process data at real-time rates. Thus, while the invention has been described in the context of communications over a wide-area network, it can be used in any type of data acquisition system. The preceding description should therefore be viewed as exemplary, rather than restrictive. The scope of the invention is indicated by the following claims, rather than the foregoing description, and all changes which come within the meaning and range of equivalents thereof are intended to be embraced therein.

We claim:

1. A signal processing system for providing a plurality of realtime services to and from a number of independent client applications and devices, said system comprising:

a subsystem comprising a host central processing unit (CPU) operating in accordance with at least one application program and a device handler program, said subsystem further comprising an adapter subsystem interoperating with said host CPU and said device;

a realtime signal processing subsystem for performing a plurality of data transforms comprising a plurality of realtime signal processing operations; and

at least one realtime application program interface (API) coupled between the subsystem and the realtime signal processing subsystem to allow the subsystem to interoperate with said realtime services.

2. The signal processing system as set forth in claim **1**, wherein said signal processing system receives and transmits a plurality of datatypes over a plurality of different wide area networks (WANs).

3. A signal processing system for providing a plurality of realtime services over a wide area network (WAN), said system comprising:

a telecommunications subsystem comprising a host central processing unit (CPU) and a wide area network interface, where said wide area network interface is comprised of a hardware interface to the network and driver software which executes on the host

a telecommunications subsystem comprising a host central processing unit (CPU) operating in accordance with at least one application program and a datastream handler program, said telecommunications subsystem further comprising a telecommunications adapter subsystem interoperating with said host CPU and said WAN;

a realtime signal processing subsystem for performing a plurality of transforms comprising a plurality of realtime signal processing operations; and

at least one realtime application program interface (API) coupled between the telecommunications subsystem and the realtime signal processing subsystem to allow the telecommunications subsystem to interoperate with said realtime services.

4. The signal processing system as set forth in claim **3**, wherein the realtime signal processing subsystem comprises:

at least one realtime communications module coupled to receive a plurality of communications commands from said applications programs via said datastream handler program and said realtime APIs, said realtime communications module in response to said communications commands issuing a plurality of requests for realtime services to at least one realtime service provider;

a translation interface program coupled to receive said requests for realtime services from said communications modules; and

a realtime processor including a realtime operating system interoperating with said translation program for executing a plurality of realtime operations comprising realtime functions in response to said requests.

5. The realtime data processing system as set forth in claim **4**, wherein the translation interface comprises a plurality of realtime features to access a modem unit for communicating over said WAN.

6. The signal processing system as set forth in claim **3** further comprising a direct memory access (DMA) unit coupled between said realtime signal processing subsystem and a hardware abstraction portion of said telecommunications subsystem, said DMA unit providing for transfer of datablocks from said telecommunications adapter module to said realtime signal processing subsystem.

7. A signal processing system for providing a plurality of realtime services over a wide area network (WAN), said system comprising:

a telecommunications subsystem comprising a host central processing unit (CPU) operating in accordance with at least one applications program and a datastream handler program, said telecommunications subsystem further comprising a telecommunications adapter subsystem interoperating with said host CPU and said WAN;

a virtual realtime device enabling a plurality of realtime signal processing operations in accordance with at least one realtime service request issued by said applications program; and

at least one realtime application program interface (API) interoperating with the telecommunications subsystem and the virtual realtime device to enable the telecommunications subsystem to interoperate with said realtime signal processing operations.

8. The signal processing system as set forth in claim **7**, wherein the virtual realtime device comprises a realtime translation interface program and virtual realtime engine, said virtual realtime engine enabling said realtime services by performing a number of data translation operations in accordance with said realtime service request and said realtime translation interface program.

9. The signal processing system as set forth in claim **8**, wherein the virtual realtime engine comprises:

a realtime processor including a realtime operating system, and

a plurality of realtime function libraries interoperatively coupled with said realtime processor for providing a plurality of processing steps comprising said realtime signal processing operations,

whereby said virtual realtime engine responds to communications commands initiated by said applications programs.

10. The signal processing system as set forth in claim 7 further comprising a direct memory access (DMA) unit coupled between said virtual realtime signal processing subsystem and a hardware abstraction portion of said telecommunications subsystem, said DMA unit providing for transfer of data from said telecommunications adapter module to said virtual realtime signal processing subsystem.

11. The signal processing system as set forth in claim 7, wherein the virtual realtime translation interface comprises a plurality of realtime features to access a modem unit for communicating over said WAN.

12. The signal processing system as set forth in claim 11, wherein the modem unit comprises at least a serial communications controller, a programmable timer, and a plurality of input/output (I/O) lines.

13. The signal processing system as set forth in claim 7, wherein realtime service requests are selected from the group of realtime service request devices consisting of telephone answering machines, automatic telephone dialing machines, and remote control systems.

14. The signal processing system as claimed in claim 6, wherein the realtime signal processing operations are selected from the group of telecommunications transactions consisting of fax (send/receive) and data transmission transactions.

15. The signal processing system as claimed in claim 14, wherein the data transmission transaction comprises at least one data framing format and at least one data protocol.

16. The virtual realtime data processing system as set forth in claim 9, wherein the realtime processor comprises a programmable processing unit which is controlled by said realtime communications applications and said realtime communications interface.

17. The virtual realtime data processing system as set forth in claim 9, wherein the realtime processor comprises said host CPU.

18. The signal processing system as set forth in claim 9, wherein the realtime processor comprises a hard digital signal processor in which the realtime operating system and the realtime function libraries are fixedly embodied in a hardware element.

19. The realtime data processing system of claim 4 comprising a plurality of realtime communications modules which are respectively associated with different realtime services.

20. The realtime data processing system of claim 19 wherein at least some of said realtime communications modules provide a service which implements a virtual realtime device.

21. The realtime data processing system of claim 19 wherein one of said realtime services comprises a video processing service.

22. The realtime data processing system of claim 19 wherein one of said realtime services comprises a sound processing service.

23. The realtime data processing system of claim 19 wherein one of said realtime services comprises a telephone service.

24. The signal processing system of claim 1, wherein said realtime signal processing subsystem comprises:

a realtime processor including an operating system for executing a plurality of realtime functions;

a realtime communications module which is independent of said realtime processor and is coupled to receive a plurality of communications commands from said application programs via said device handler program and said realtime API, said realtime communications module operating in response to said communications commands to issue a plurality of requests for realtime services to said realtime processor; and

a translation interface program which is specific to said realtime processor and is coupled to receive said requests for realtime services from said communications module and provide said requests to said realtime processor.

25. The signal processing system of claim 24 comprising a plurality of realtime communications modules which are respectively associated with different realtime services.

26. The signal processing system of claim 25 wherein at least some of said realtime communications modules provide a service which implements a virtual realtime device.

27. The signal processing system as set forth in claim 24, wherein the realtime processor comprises a hard digital signal processor in which said operating system and realtime function libraries are fixedly embodied in a hardware element.

28. The signal processing system as set forth in claim 24, wherein the realtime processor comprises said host CPU.

29. The signal processing system of claim 24, wherein said realtime processor is embodied in a hardware device and includes realtime function libraries that are embodied in programmable software.

30. The signal processing system of claim 29 wherein said operating system is also embodied in programmable software.

31. A signal processing system, comprising:

an input/output device for sending and/or receiving isochronous streams of data transmitted over a communications path;

a realtime engine for performing data transformations on the isochronous streams of data, said realtime engine being independent of said input/output device;

a device handler program associated with said input/output device, for generating requests to the realtime engine to perform data transformations on the isochronous streams of data; and

at least one application programming interface for receiving the requests generated by said device handler program and issuing commands to said realtime engine to perform the requested data transformations.

32. The signal processing system of claim 31 comprising a plurality of said application programming interfaces which are respectively associated with different types of services to be provided by said realtime engine with respect to isochronous streams of data.

33. The signal processing system of claim 32 wherein one of said application programming services relates to the operation of the realtime engine as a virtual telephone device.

34. The signal processing system of claim 32 wherein one of said application programming services relates to the operation of the realtime engine as a virtual sound device.

35. The signal processing system of claim 32 wherein one of said application programming services relates to the operation of the realtime engine as a virtual video device.

36. The signal processing system of claim 31 wherein said application programming interface includes command/control registers that are shared between said realtime engine and said device handler program for transferring said requests and responding thereto, and a buffer for transferring isochronous streams of data between said device handler program and said realtime engine.

37. The signal processing system of claim 31, wherein said realtime engine comprises:

a realtime processor including an operating system for executing a plurality of realtime functions;

a communications module which is independent of said realtime processor and is coupled to receive said commands from said application programming interface, said communications module operating in response to said commands to issue a plurality of requests for realtime services to said realtime processor; and

a translation interface program which is specific to said realtime processor and is coupled to receive said requests for realtime services from said communications module and provide said requests to said realtime processor.

38. The signal processing system as set forth in claim **37**, wherein the realtime processor comprises a hard digital signal processor in which said operating system and realtime function libraries are fixedly embodied in a hardware element.

39. The signal processing system as set forth in claim **37**, wherein said processing system is incorporated in a data processing system having a host central processing unit (CPU), and wherein the realtime processor comprises said host CPU operating in accordance with software instructions relating to said realtime functions.

40. The signal processing system of claim **37**, wherein said realtime processor is embodied in a hardware device and includes realtime function libraries that are embodied in programmable software.

41. The signal processing system of claim **40** wherein said operating system is also embodied in programmable software.

* * * * *