EXHIBIT

33

US005502839A

United States Patent [19]

Kolnick

[11] **Patent Number: 5,502,839**

[45] **Date of Patent: Mar. 26, 1996**

[54] **OBJECT-ORIENTED SOFTWARE ARCHITECTURE SUPPORTING INPUT/OUTPUT DEVICE INDEPENDENCE**

[75] Inventor: **Frank C. Kolnick**, Willowdale, Canada

[73] Assignee: **Motorola, Inc.**, Schaumburg, Ill.

[21] Appl. No.: **361,738**

[22] Filed: **Jun. 2, 1989**

Related U.S. Application Data

[63] Continuation of Ser. No. 619, Jan. 5, 1987, abandoned.

[51] **Int. Cl.**^{6} **G06F 13/00**

[52] **U.S. Cl.** **395/800**; 364/228.2; 364/237.9; 364/239.9; 364/280; 364/284.2; 364/DIG. 1

[58] **Field of Search** .... 364/200 MS File, 364/900 MS File; 395/500

[56] **References Cited**

U.S. PATENT DOCUMENTS

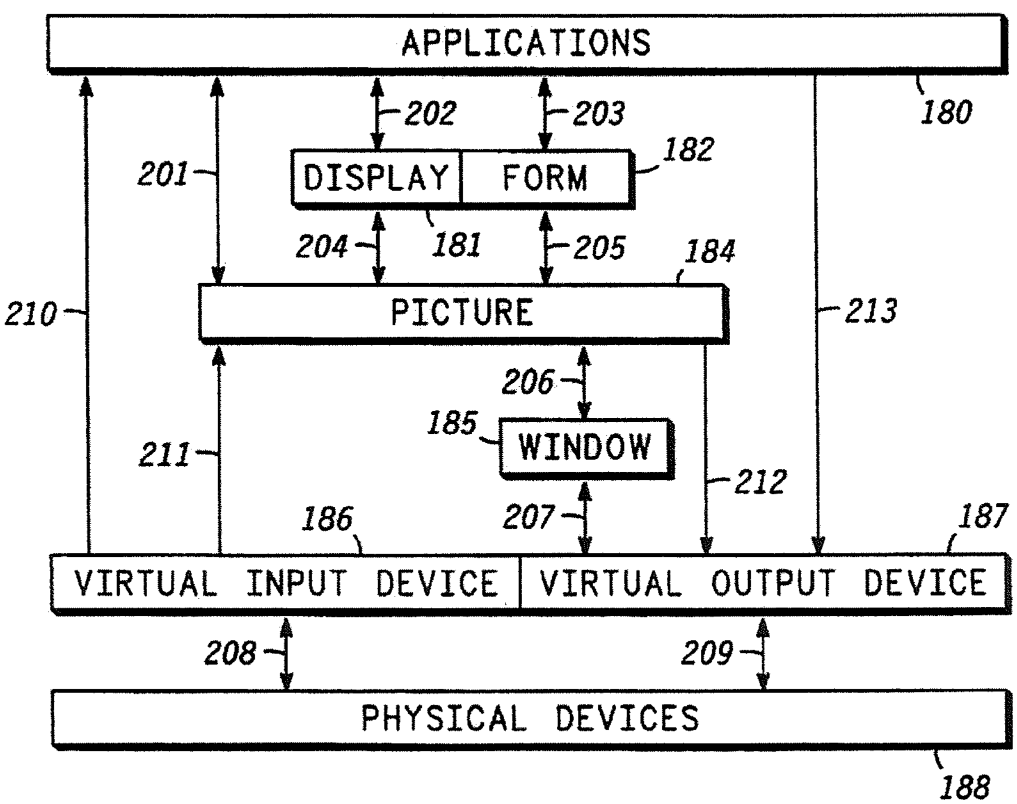
3,930,232	12/1975	Wallach et al.	395/500
4,241,341	12/1980	Thorson	340/747
4,454,593	6/1984	Fleming	364/900
4,485,439	11/1984	Rothstein	395/500
4,547,628	10/1985	Tamura et al.	340/747
4,555,775	11/1985	Pike	364/900
4,559,614	12/1985	Peek et al.	364/900
4,642,790	2/1987	Minshull et al.	364/900
4,754,395	6/1988	Weisshaar et al.	364/200
4,800,523	1/1989	Gerety et al.	395/500
4,858,114	8/1989	Heath et al.	395/500
5,063,494	11/1991	Davidowski et al.	395/800

Primary Examiner—Kevin J. Teska
Assistant Examiner—Ayni Mohamed
Attorney, Agent, or Firm—Walter W. Nielsen; Harold C. McGurk; S. Kevin Pickens

[57] **ABSTRACT**

An object-oriented software architecture interacts with "real" input/output devices exclusively through "virtual" input/output devices. Since all human interface with the operating system is performed through such virtual devices, the system can accept any form of real input or output devices. The lowest level of the operating system converts input from any physical device to virtual form and converts virtual output into suitable physical output. Any number of physical devices can be connected to, removed from, or replaced in the system without disrupting the system.

23 Claims, 9 Drawing Sheets



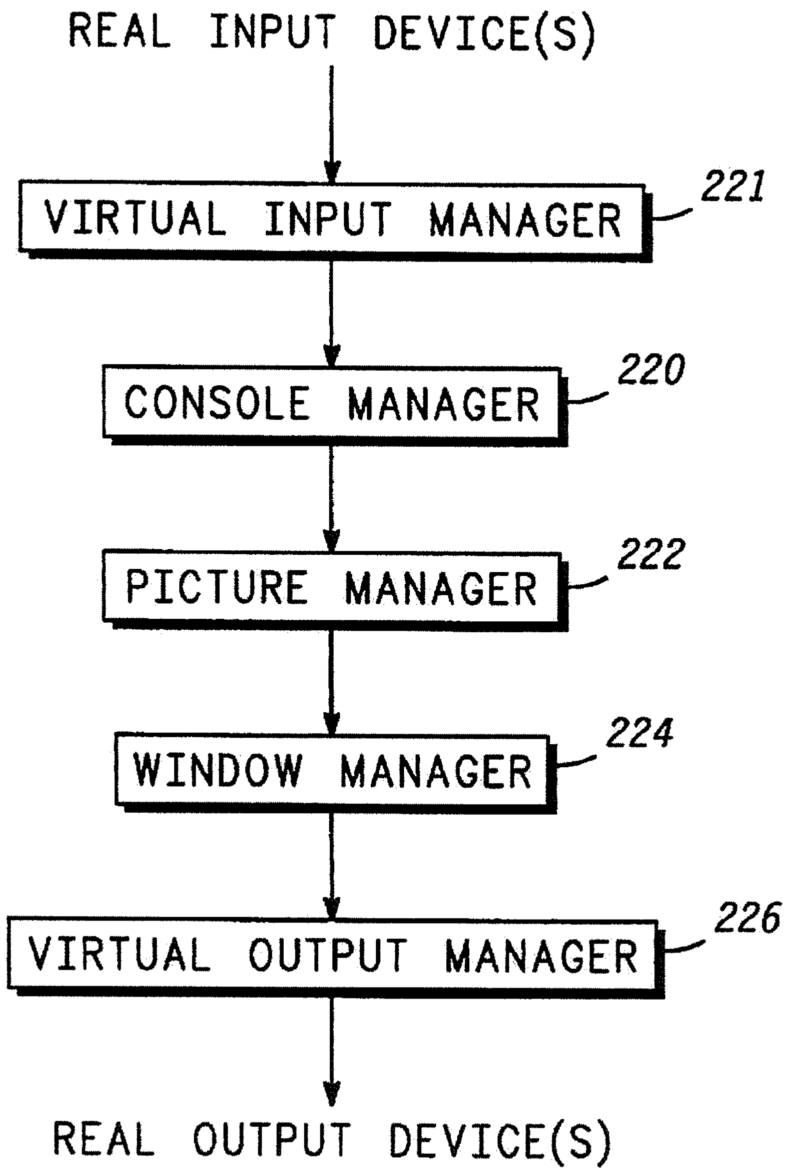

REAL INPUT DEVICE(S)
VIRTUAL INPUT MANAGER
221
CONSOLE MANAGER
220
PICTURE MANAGER
222
WINDOW MANAGER
224
VIRTUAL OUTPUT MANAGER
226
REAL OUTPUT DEVICE(S)

FIG. 12

OBJECT-ORIENTED SOFTWARE ARCHITECTURE SUPPORTING INPUT/OUTPUT DEVICE INDEPENDENCE

This application is a continuation of prior application Ser. No. 000,619, filed Jan. 5, 1987 now abandoned.

RELATED INVENTIONS

The present invention is related to the following inventions, all filed on May 6, 1985, and all assigned to the assignee of the present invention:

1. Title: Nested Contexts in a Virtual Single Machine
Inventors: Andrew Kun, Frank Kolnick, Bruce Mansfield
Ser. No.: 730,903, now abandoned
2. Title: Computer System With Data Residence Transparency and Data Access Transparency
Inventors: Andrew Kun, Frank Kolnick, Bruce Mansfield
Ser. No.: 730,929 (now abandoned) and Ser. No. 07/110,614 filed on Oct. 14, 1987 and now abandoned (continuation)
3. Title: Network Interface Module With Minimized Data Paths
Inventors: Bernhard Weisshaar, Michael Barnea
Ser. No.: 730,621, now U.S. Pat. No. 4,754,395
4. Title: Method of Inter-Process Communication in a Distributed Data Processing System
Inventors: Bernhard Weisshaar, Andrew Kun, Frank Kolnick, Bruce Mansfield
Ser. No.: 730,892, now U.S. Pat. No. 4,694,396
5. Title: Logical Ring in a Virtual Single Machine
Inventor: Andrew Kun, Frank Kolnick, Bruce Mansfield
Ser. No.: 730,923 (now abandoned) Ser. No. 183,469 filed on Apr. 13, 1988 and now U.S. Pat. No. 5,047,925 (continuation)
6. Title: Virtual Single Machine With Message-Like Hardware Interrupts and Processor Exceptions
Inventors: Andrew Kun, Frank Kolnick, Bruce Mansfield
Ser. No.: 730,922 now U.S. Pat. No. 4,835,685

The present invention is also related to the following inventions, all filed on even date herewith, and all assigned to the assignee of the present invention:

7. Title: Computer Human Interface Comprising User-Adjustable Window for Displaying or Printing Information
Inventor: Frank Kolnick
Ser. No.: 000,625 now abandoned
8. Title: Computer Human Interface With Multi-Application Display
Inventor: Frank Kolnick
Ser. No.: 000,620 now abandoned
9. Title: Self-Configuration of Nodes in a Distributed Message-Based Operating System
Inventor: Gabor Simor
Ser. No.: 000,621 now U.S. Pat. No. 5,165,018
10. Title: Process Traps in a Distributed Message-Based Operating System
Inventors: Gabor Simor
Ser. No.: 000,624 now abandoned
11. Title: Computer Human Interface With Multiple Independent Active Pictures and Windows
Inventor: Frank Kolnick
Ser. No.: 000,626, now abandoned

TECHNICAL FIELD

This invention relates generally to digital data processing, and, in particular, to a human interface system providing means for converting "real" input into virtual input, and means for converting virtual output into "real" output.

BACKGROUND OF THE INVENTION

It is known in the data processing arts to couple a wide assortment of input and output devices to a data processing system for the purpose of providing an appropriate human interface. Such devices may take the form of keyboards of varying manufacture, "mice", touch-pads, joy-sticks, light pens, video screens, audio-visual signals, printers, etc.

Due to the wide variety of I/O devices which can be utilized in the human/computer interface, it would be very desirable to isolate the human interface software from specific device types. The I/O should be independent of any particular "real" devices.

There is thus a need for a computer human interface which performs I/O operations in an abstract sense, independent of particular "real" devices.

BRIEF SUMMARY OF INVENTION

Accordingly, it is an object of the present invention to provide a data processing system having an improved human interface.

It is also an object of the present invention to provide an improved human interface system which performs input/output operations in an abstract sense, independent of any particular I/O devices.

It is another object of the present invention to provide an improved human interface system in which any type of "real" input and output devices may be employed, and in which I/O devices may be connected to and disconnected from the data processing system without disrupting processing operations.

These and other objects are achieved in accordance with a preferred embodiment of the invention by providing a virtual input interface in a data processing system, such interface comprising means for accepting input from at least one physical device, means for converting the physical device input into virtual input, and means responsive to the virtual input for performing processing operations upon the virtual input.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention is pointed out with particularity in the appended claims. However, other features of the invention will become more apparent and the invention will be best understood by referring to the following detailed description in conjunction with the accompanying drawings in which:

FIG. 1 shows a representational illustration of a single network, distributed message-based data processing system of the type incorporating the present invention.

FIG. 2 shows a block diagram illustrating a multiple-network, distributed message-based data processing system of the type incorporating the present invention.

FIG. 3 shows an architectural model of a data processing system of the type incorporating the present invention.

FIG. 4 shows the relationship between software contexts and processes as they relate to the present invention.

FIG. 5 shows how messages may be sent between processes within nested contexts.

FIG. 6 shows a standard message format used in the distributed data processing system of the present invention.

FIG. 7 shows the relationship between pictures, views, and windows in the human interface of a data processing system of the type incorporating the present invention.

start-up. These processes then start up their respective subsystems. A node configuration service on each node sends configuration messages to each subsystem when it is being initialized, informing it of the devices it owns. Thereafter, similar messages are sent whenever a new device is added to the node or a device fails or is removed from the node.

Thus there is no well-defined meaning for "system up" or "system down"—as long as any node is active, the system as a whole may be considered to be "up". Nodes can be shut down or started up dynamically without affecting other nodes on the network. The same principle applies, in a limited sense, to peripherals. Devices which can identify themselves with regard to type, model number, etc. can be added or removed without operator intervention.

FIG. 6 shows the standard format of a message in a distributed data processing system of the type incorporating the present invention. The message format comprises a message i.d. portion **150**; one or more "triples" **151**, **153**, and **155**; and an end-of-message portion **160**. Each "triple" comprises a group of three fields, such as fields **156**–**158**.

The first field **156** of "triple" **151**, designated the PCRT field, represents the name of the process to be created. The second field **157** of "triple" **151** gives the size of the data field. The third field **158** is the data field.

The first field **159** of "triple" **153**, designated the PNTF field, represents the name of the process to notify when the process specified in the PCRT field has been created.

A message can have any number of "triples", and there can be multiple "triples" in the same message containing PCRT and PNTF fields, since several processes may have to be created (i.e. forming a context, as described hereinabove) for the same resource.

As presently implemented, portion **150** is 16 bytes in length, field **156** is 4 bytes, field **157** is 4 bytes, field **158** is variable in length, and EOM portion **160** is 4 bytes.

HUMAN INTERFACE—GENERAL

The Human Interface of the present invention provides a set of tools with which an end user can construct a package specific to his applications requirements. Such a package is referred to as a "metaphor", since it reflects the user's particular view of the system. Multiple metaphors can be supported concurrently. One representative metaphor is, for example, a software development environment.

The purpose of the Human Interface is to allow consistent, integrated access to the data and functions available in the system. Since users' perceptions of the system are based largely on the way they interact with it, it is important to provide an interface with which they feel comfortable. The Human Interface allows a systems designer to create a model consisting of objects that are familiar to the end user and a set of actions that can be applied to them.

The fundamental concept of the Human Interface is that of the "picture". All visually-oriented information, regardless of interpretation, is represented by pictures. A picture (such as a diagram, report, menu, icon, etc.) is defined in a device-independent format which is recognized and manipulated by all programs in the Human Interface and all programs using the Human Interface. It consists of "picture elements", such as "line", "arc", and "text", which can be stored compactly and transferred efficiently between processes. All elements have common attributes like color and fill pattern. Most also have type-specific attributes, such as typeface and style for text. Pictures are drawn in a large "world" co-ordinate system composed of "virtual pixels".

Because all data is in the form of pictures, segments of data can be freely copied between applications, e.g., from a live display to a word processor. No intermediate format or conversion is required. One consequence of this is that the end user or original equipment manufacturer (OEM) has complete flexibility in defining the formats of windows, menus, icons, error messages, help pages, etc. All such pictures are stored in a library rather than being built into the software and so are changeable at any time without reprogramming. A comprehensive editor is available to define and modify pictures on-line.

All interaction with the user's environment is through either "virtual input" or "virtual output" devices. A virtual input device accepts keyboards, mice, light pens, analog dials, pushbuttons, etc. and translates them into text, cursor-positioning, action, dial, switch, and number messages. All physical input devices must map into this set of standard messages. Only one process, an input manager for the specific device, is responsible for performing the translation. Other processes can then deal with the input without being dependent on its source.

Similarly, a virtual output manager translates standard output messages to the physical representation appropriate to a specific device (screen, printer, plotter, etc.) A picture drawn on any terminal or by a process can be displayed or printed on any device, subject to the physical limitations of that device.

With reference to FIG. 7, two "pictures" are illustrated picture A (**170**) and picture B (**174**).

The concept of a "view" is used to map a particular rectangular area of a picture to a particular device. In FIG. 7, picture A is illustrated as containing at least one view **171**, and picture B contains at least one view **175**. Views can be used, for example, to partition a screen for multiple applications or to extract page-sized subsets of a picture for printing.

If the view appears on a screen it is contained in a "window". With reference again to FIG. 7, view **171** of picture A is mapped to screen **176** as window **177**, and view **175** of picture B is mapped as window **178**.

The Human Interface allows the user to dynamically change the size of the window, move the window around on the screen, and move the picture under the window to view different parts of it (i.e., scroll in any direction). If a picture which is mapped to one or more windows changes, all affected views of that picture on all screens are automatically updated. There is no logical limit to the number or sizes of windows on a particular screen. Since the system is distributed, it's natural for pictures and windows to be on different nodes. For example, several alarm displays can share a single, common picture.

The primary mechanism for interacting with the Human Interface is to move the cursor to the desired object and "select" it by pressing a key or button. An action may be performed automatically upon selection or by further interaction, often using menus. For example, selecting an icon usually activates the corresponding application immediately. Selecting a piece of text is often followed by selection of a command such as "cut" or "underline" Actions can be dynamically mapped to function keys on a keyboard so that pressing a key is equivalent to selecting an icon or a menu item. A given set of cursors (the cursor changes as it moves from one application picture to another), windows, menus, icons, and function keys define a "metaphor".

The Human Interface builds on the above concepts to provide a set of distributed services. These include electronic mail, which allows two or more users at different terminals to communicate with each other in real time or to queue files for later delivery, and a forms manager for data entry. A subclass of windows called "virtual terminals" provides emulation of standard commercially available terminals.

FIG. **8** shows the different levels of the Human Interface and data flow through them. Arrows **201–209** indicate the most common paths, while arrows **210–213** indicate additional paths. The interface can be configured to leave out unneeded layers for customized applications. The philosophy behind the Human Interface design dictates one process per object. That is, a process is created for each active window, picture, input or output device, etc. As a result, the processes are simplified and can be distributed across nodes almost arbitrarily.

MULTIPLE INDEPENDENT PICTURES AND WINDOWS

A picture is not associated with any particular device, and it is of virtually unlimited size. A "window" is used to extract a specified rectangular area—called a "view"—of picture information from a picture and pass this data to a virtual output manager.

The pictures are completely independent of each other. That is, none is aware of the existence of any other, and any picture can be updated without reference to, and without affect upon, any other. The same is true of windows.

Thus the visual entity seen on the screen is really represented by two objects: a window (distinguished by its frame title, scroll bars, etc.), and a picture, which is (partially) visible within the boundaries of the window's frame.

As a consequence of this autonomy, multiple pictures can be updated simultaneously, and windows can be moved around on the screen and their sizes changed without the involvement of other windows and/or pictures.

Also, such operations are done without the involvement of the application which is updating the window. For example, if the size of a window is increased to look at a larger area of the picture, this is handled completely within the human interface.

HUMAN INTERFACE—PRIMARY FEATURES

The purpose of the Human Interface is to transform machine-readable data into human-readable data and vice versa. In so doing the Human Interface provides a number of key services which have been integrated to allow users to interact with the system in a natural and consistent manner. These features will now be discussed.

Device Independence—The Human Interface treats all devices (screens, printers, etc.) as "virtual devices" None of the text, graphics, etc. in the system are tied to any particular hardware configuration. As a result such representations can be entered from any "input" device and displayed on any "output" device without modification. The details of particular hardware idiosyncracies are hidden in low-level device managers, all of which have the same interface to the Human Interface software.

Picture Drawing—The Human Interface can draw "pictures" composed of any number of geometric elements, such as lines, circles, rectangles, etc., as well as any arbitrary shape defined by the user. Each element can have its own color and line thickness. In addition, closed figures may be filled in with a particular shading pattern in any given color. A picture can be of almost any size. All output from the Human Interface to a user is via pictures, and all input from a user to the Human Interface is stored as pictures, so that there is only one representation of data within the Human Interface.

Text can be freely intermixed with graphical images, so that the user need only learn one "editor" to do his job. Consequently it is not necessary to switch between editors or "cut and paste" between pictures. Text characters can be selected from a large predefined character set, which includes mathematical and Greek symbols, among others, and can be typed in a wide variety of fonts, colors, sizes, and styles (e.g. bold, italic, or underlined). It is also possible for a user to define his own symbols and add them to the character set.

Windowing—The Human Interface allows the user to partition a screen into as many "sub-screens" or "windows" as required to view the information he desires. The Human Interface places no restrictions on the contents of such windows, and all windows can be simultaneously updated in real time with data from any number of concurrently executing programs. Any picture can be displayed, created, or modified ("edited") in any window. Also any window can be expanded or contracted, or it can be moved to a new location on the screen at any time.

If the current picture is larger than the current window, the window can be scrolled over the picture, usually in increments of a "line" or a "page". It is also possible to temporarily expand or contract the visible portion of the picture ("zoom in" or "zoom out") without changing the window's dimensions and without changing the actual picture.

Dialog Management—The Human Interface is independent of any particular language or visual representation. That is, there are no built-in titles, menus, error messages, help text, icons, etc. for interacting with the system. All such information is stored as pictures which can be modified to suit the end user's requirements, either prior to or after installation. The user can modify the supplied dialog with his own at any time.

Data Entry—The Human Interface provides a generalized interface between the user and any program (such as a data base manager) which requires data from the user. The service is called "forms management", because a given data structure is displayed as a fill-in-the-blanks type of "form" consisting of numerous modifiable fields with descriptive labels. The Human Interface form is interactive, so that data can be verified as it is entered, and the system can assist the user by displaying explanatory text when appropriate (on demand or as a result of an error).

Communication Between Users—The Human Interface permits two or more users to "converse" with each other in real time or to send "mail" to each other. Conversation is performed through a window on each of the user's screens. Mail is sent by creating a picture (text and/or diagrams) and specifying a destination. The destination may be one particular user, a group of users, or all users in the system (i.e. a "broadcast"). Transmission may be immediate or delayed until a given date and time or until the given user(s) sign onto the system. When mail arrives at the destination, the receiving user is informed and may then read, save, print, or erase the picture.

Event Management—The Human Interface can record any arbitrary event for future reference. The Human Interface defines a simple, yet flexible grammar for forming problems of its own (e.g. "printer out of paper"). Icons are displayed by the Console Manager automatically when a specific frame of reference is requested by the user. The Console Manager may also display informational messages (such as "console starting up") which are automatically erased when the associated action is finished.

Picture Manager—It is created when a picture is built, and it exits when the picture is no longer required. There is one Picture Manager per picture. The Picture Manager constructs a device-independent representation of a picture using a small set of elemental "picture elements" and controls modification and retrieval of the elements.

A Picture Manager knows about the following processes: the process which created it, and the Draw Manager. The following processes know about the Picture Manager: the Console Manager in the same context, and Window Managers in the same context.

A Picture Manager is created to handle exactly one picture, and it need only be created when that picture is being accessed. It can be told to quit at any time, deleting its representation of the picture. Some other process must copy the picture to a file if it needs to be saved.

When a Picture Manager first starts up, its internal picture is empty. It must receive a "load file" request, or a series of "draw" requests, before a picture is actually available. Until that is done any requests which refer to specific elements or locations in the picture will receive an appropriate "not found" status message.

A picture is logically composed of device-independent "elements", such as text, line, arc, and symbol. In general, there is a small number of such elements. Each element consists of a common header, which includes the element's position in the picture's coordinate system, its color, size, etc., and a "value" which is unique to the element's type (e.g. a character string, etc.). The header also specifies how the element combines with other elements in the picture (overlays them, merges with them, etc.). A special element type called "null" is also supported to facilitate the removal of picture elements from pictures or other similar large lists without forcing time-consuming compaction procedures. Any element can therefore be redefined to "null", indicating that it should be ignored for all future processing.

The "null" color (zero) is treated as transparent when used in either the foreground or the background. Specifically, if the foreground color is null, the element itself is not drawn, but it may still be filled in. If the background color is null, the element is not filled in. If the shading pattern is null, and the color is not null, the background fill is solid.

A picture is represented in an internal format which may be different from the external representation of picture elements and which is, in any case, hidden from other processes. This representation is designed to optimize retrieval of picture elements, with a secondary emphasis on adding new elements and modifying or erasing old ones. The order in which the elements were originally drawn is preserved (unless explicit "order" requests have been received to re-arrange them).

Requests to "animate" an element result in the creation of a separate, local "animate" process which performs the necessary transformations and sends the appropriate requests (usually "draw" or "erase") back to the Picture Manager periodically.

A Picture Manager processes incoming requests one at at time, as it receives them. Each message can change the state of the picture for later requests. The Picture Manager supports numerous operations, including the following: "draw" new elements; "modify", "overwrite", or "erase" existing elements; "copy" or "move" elements to another location in the same picture or to any other given process; "group" elements together into one (or "ungroup" them); "scale" them (i.e. expand, stretch, or shrink them); and "rotate" them. It can also be asked to "notify" a particular process if any elements within a given rectangular area (the "viewport") are changed and to determined whether a given location coincides (or come close to) any element in the picture. Any response to a request (e.g., multiple picture elements) is sent in a single message.

When an element is sent as the result of an outstanding "notify" request, all elements which overlap it (and all elements which overlap those elements) are sent as well. These are sent together in one message. The background is displayed by generating a "rectangle" element of the same size as the current viewport with a null foreground color and the appropriate background pattern and color. This element is always the lowest level in the picture; i.e., it is sent before all others. All erasure of elements from a display is accomplished by "draw" requests which redisplay the background and/or elements in the picture, overwriting the "erased" elements. There is no explicit "erase" request to a window (or output) manager.

Input Manager—There is one Input Manager per set of "logical input devices" (such as keyboards, mice, light pens, etc.) connected to the system. The Input Manager handles input interrupts and passes them to the console manager. Cursor movement inputs may also be sent to a designated output manager.

The Input Manager knows about the following processes: the process which initialized it, and possibly one particular Output Manager in the same context. The following process knows about the Input Manager: the Console Manager in the same context.

An Input Manager is created (automatically, at system start-up) for each set of "logical input devices" in the system, thus implementing a single "virtual keyboard" There can only be one such set, and therefore one Input Manager, per Console context. The software (message) interface to each manager is identical, although their internal behavior is dependent upon the physical device(s) to which they communicate. All input devices interrupt service routines (including mouse, digitizing pad, etc.) are contained in Input Managers and hidden from other processes. When ready, each Input Manager must send an "I'm here" message to the closest process named "Console".

An Input Manager must be explicitly initialized and told to proceed before it can begin to process input interrupts. Both of these are performed using appropriate messages. Whichever process initializes the manager becomes tightly coupled to it, i.e., they can exchange messages via PID's rather than by name. The Input Manager will send all inputs to this process (usually the Console Manager). This coupling cannot be changed dynamically; the manager would have to be re-initialized. Between the "initialize" and the "proceed" an Input Manager may be sent one or more "set" requests to define its behavior. It does not need to be able to interpret the meaning of any input beyond distinguishing cursor from non-cursor. Device-independent parameters (such as pixel size and density) are not down-loaded but rather are assumed to be built into the software, some part of which, in general, must be unique to each type of Input Manager.

An Input Manager can be dynamically "linked" to a particular Output Manager, if desired. If so, all cursor control input (or any other given subset of the character set) will be sent to that manager, in addition to the initializing process, as it is received. This assignment can be changed or cut off at any time. (This is generally useful only if the output device is a screen.)

In general, input is sent as single "characters", each in a single "K" (i.e. keyboard string) message (unbuffered) to the specified process(es). Some characters, such as "shift one" or a non-spacing accent, are temporarily buffered until the next character is typed and are then sent as a pair. Redefinable characters, including all displayable text, cursor control commands, "action keys", etc. are sent as triples.

New output devices can be added to the "virtual keyboard" at any time by re-initializing the manager and down-loading the appropriate parameters, followed by a "proceed". All input is suspended while this is being done. Previously down-loaded parameters and the screen assignment are not affected. Similarly, devices can be disconnected by terminating (sending "quit" requests for) them individually. A nonspecific "quit" terminates the entire manager.

Where applicable, an Input Manager will support requests to activate outputs on its device(s), such as lights or sound generators (e.g., a bell).

The Input Process is a distinct process which is created by each Console Manager for its Input Manager to keep track of the current input state. In general, this includes a copy of its last input of each type (text, function key, pointer, number, etc.), the current redefinable character set number, as well as Boolean variables for such conditions as "keyboard locked", "select key depressed" (and being held down), etc. The process is simply named "In". The Input Manager is responsible for keeping this process up-to-date. Any process may examine (but not modify) the contents of "In".

Output Manager—There is one Output Manager per physical output device (screen, printer, plotter, etc.) connected to the system. Each Output Manager converts (and possibly scales) standard "pictures" into the appropriate representation on its particular device.

The Output Manager knows about the following processes: the process which initialized it, and the Draw Manager in the same context. The following processes know about the Output Manager: the Console Manager in the same context, the Input Manager in the same context, and the Window Manager in the same context.

An Output Manager is created (automatically, at system start-up) for each physical output device in the system, thus implementing numerous "virtual screens". There can be any number of such devices per Console context. The software (message) interface to each manager is identical, although their internal behavior is dependent upon the physical device(s) to which they communicate. All output interrupt service routines (if any) are contained in Output Manager and hidden from other processes. Each manager also controls a process called Cursor which holds information concerning its own cursor. When ready, each Output Manager must send an "I'm here" message to the closest process named "Console".

An Output Manager must be explicitly initialized and told to proceed before it can begin to actually write to its device. Both of these are performed using appropriate Human Interface messages. Which process initializes the manager becomes tightly coupled to it; i.e., they can exchange messages via PID's rather than by name. This coupling cannot be changed dynamically; the manager would have to be re-initialized. Between the "initialize" and the "proceed" an Output Manager may be sent one or more "set" requests to define its behavior. Device-independent parameters (such as pixel size and density) are not down-loaded but rather are assumed to be built into the software, some part of which, in general, must be unique to each type of Output Manager. Things like a screen's background color and pattern are down-loadable at start-up time and at any other time.

In general, an Output Manager is driven by "draw" commands (containing standard picture elements) sent to it by any process (usually a Window Manager). Its primary function then is to translate picture elements, described in terms of virtual pixels, into the appropriate sequences of output to its particular device. It uses the Draw Manager to expand elements into sets of real pixels and keeps the Cursor process informed of any resulting changes in cursor position. It looks up colors and shading patterns in predefined tables. The "null" color (zero) is interpreted as "draw nothing" whenever it is encountered. A "clear" request is also supported. It changes a given polygonal area to the screen's default color and shading pattern.

Any "draw" request can be preceded by a "clip" request. "Clip" means "don't display pixels outside of given polygon", i.e. only the logical AND of the polygonal area and the given picture elements is drawn. The clip request applies only to the next draw request received from the same process and is then discarded.

"Text" elements are displayed by the output device's built-in character generator, if possible. However, most text is created from predefined bit-maps which are stored in a Human Interface library. Different bit-maps exist for various combinations of font and size. Sizes which are not explicitly stored must be calculated from the available bit-maps when required. The style is always generated dynamically, i.e., it is calculated from the basic bit-map.

Output Managers also accept "K" messages (i.e. keyboard strings) containing cursor movement commands. If the associated device is a screen, the manager erases the cursor from its current position (if necessary, i.e. if the cursor is not supported directly by the hardware) and redraws it in its new location. It uses the Cursor Process to get a symbol element representing the cursor's current shape and color, and it tells it the new location after it has redrawn the cursor. (The manager may have to ask its initializing process to redraw the part of the picture which was previously obscured by the cursor after it moves it.) If the associated device is not a real screen, cursor movement commands are simply ignored.

If possible, an Output Manager should be able to save, restore, move, and copy rectangular areas of the virtual screen. These are primarily speed-optimizing operations, and they need not always be supplied. In general, an Output Manager can be queried for its characteristics, e.g., whether it supports the above functions, whether it is bit-mapped or character-oriented, the output dimensions (in pixels or characters, as appropriate), the physical size, etc.

The Cursor Process is a distinct process which is created by each Console Manager in its context to keep track of the cursor. That process, which has the same name as the screen (not the Output Manager), knows the current location of the cursor, all of the symbols which may represent the cursor on the screen, which symbol is currently being used, how many real pixels to move when a cursor movement command is executed, etc. It can, in general, be accessed for any of this information at any time by any process. The associated Output Manager is the prime user of this process and is responsible for keeping it up to date. The associated Input Manager (if any) is the next most common user, requesting the cursor's position every time it processes a "command" input.

Dialog Manager—There is one Dialog Manager per console, and it provides access to a library of "pictures" which define the menus, help texts, prompts, etc. for the Human as "select key is being held down", "keyboard locked", etc. One input process is created by each Console Manager. The console's input message updates the process; any other process may query it.

The Human Interface is structured as a collection of subsystems, implemented as contexts, each of which is responsible for one broad area of the interface. There are two major contexts accessible from outside the Human Interface: "Console" and "Print" They handle all screen/keyboard interaction and all hard-copy output, respectively. These contexts are not necessarily unique. There may be one or more instances of each in the system, with possibly several on the same cell. Within each, there may be several levels of nested contexts.

The possible interaction between various Human Interface components will now be described.

Console Manager/Other Contexts—Processes of other contexts may send requests for console services or notification of relevant events directly to the Console Manager(s). The Console Manager routes messages to the appropriate service. It also notifies (via a "status" message) the current owner of a window whenever an object in its window has been selected. Similarly, it sends a message to an application when a user requests that application in a particular window.

Console Manager/Input Manager—The Console Manager initializes the Input Manager and usually assigns a particular Output Manager to it. The Input Manager always sends all input (one character, one key, one cursor movement, etc. at a time) directly to the Console Manager. It may also send "status" messages, either in response to a "download", "initialize", or "terminate" request, or any time an anomaly arises.

Console Manager/Output Manager—The Console Manager displays information on its "prime" output device during system start-up and shut-down without using pictures and windows. It therefore sends picture elements directly to an Output Manager. The Console Manager is also responsible for moving the cursor on the screen while the system is running, if applicable. The Console Manager (or any other Human Interface manager, such as an "editor") may change the current cursor to any displayable symbol. Output Managers will send "status" messages to the Console Manager any time an anomaly arises.

Console Manager/Picture Manager—The Console Manager creates Picture Managers on demand and tells each of them the name of a file which contains picture elements, if applicable. A Picture Manager can also accept requests from the Console Manager (or anyone else) to add elements to a picture individually, delete elements, copy them, move them, modify their attributes, or transform them. It can be queried for the value of an element at (or close to) a given location within its picture. The Console Manager will tell a Picture Manager to erase its picture and exit when it is no longer needed. A Picture Manager usually sends "status" messages to the Console Manager whenever anything unusual (e.g., an error) occurs.

Console Manager/Window Manager—The Console Manager creates Window Managers on demand. Each Window Manager is told its size, the PID of an Output Manager, the coordinates (on the screen) of its upper left outside corner, the characteristics of its frame, the PID of a particular Picture Manager, the coordinates of the first element from which to start displaying the picture, and the name of the process which "owns" the window. While a window is active, it can be requested to re-display the same picture starting at a different element or to display a completely different picture.

The coordinates of the window itself may be changed, causing it to move on the screen, or it may be told to change its size, frame, or owner. A Window Manager can be told to "clip" the picture elements in its display along the edges of a given polygon (the default polygon is the inside edge of the window's frame). It can also be queried for the element corresponding to a given coordinate. The Console Manager will tell a Window Manager to "close" (erase) its window and exit when it is no longer needed. A Window Manager sends "status" messages to the Console Manager to indicate success or failure of a request.

Console Manager/Dialog Manager—The Dialog Manager accepts requests to load and/or dynamically create "pictures" which represent menus, prompts, error messages, etc. In the case of interactive pictures (such as menus), it also interprets the response for the Console Manager. Other processes may also use the Dialog Manager through the Console Manager.

Console Manager/Prime Manager—Console Managers generally send "spool" requests to Print Managers to get hard-copies of screens or pictures. An active picture must first be copied to a file. The Print Manager returns a "status" message when the request is complete or if it fails.

Window Manager/Picture Manager—A Window Manager requests lists of one or more picture elements from the relevant Picture Manager, specified by the coordinates of a rectangular "viewport" in the picture. It can also request the Picture Manager to automatically send changes (new, modified, or erased elements), or just notification of changes, to it. The Picture Manager sends "status" messages to notify the Window Manager of changes or errors.

Window Manager/Output Manager—A Window Manager sends lists of picture elements to its Output Manager, prefixed by the coordinates of a polygon by which the Output Manager is to "clip" the pixels of the elements as it draws them. A given list of picture elements can also be scaled by a given factor in any of its dimensions. The Output Manager returns a "status" message when a request fails.

Input Manager/Output Manager—The Input Manager sends all cursor movement inputs to a pre-assigned Output Manager (if any), as well as to the Console Manager. This assignment can be changed dynamically.

Print Manager/Other Processes—The Print Manager accepts requests to "spool" a file or to "print" one or more picture elements. It sends a "status" message at the completion of the request or if the request cannot be carried out. The status of a queued request can also be queried or changed at any time.

Print Manager/File Manager—The Print Manager reads picture elements from a File Manager (whose name was sent to it via a "spool" request). It may send a request to "delete" the file back to the File Manager after it has finished printing the picture.

Print Manager/Picture Manager—A Print Manager creates a Picture Manager for each spooled picture that it is currently printing, giving it the name of the relevant file. It then requests "pages" of the picture (depending upon the characteristics of the output device) one at a time. Finally, it tells the Picture Manager to go away.

Print Manager/Output Manager—The Print Manager sends picture elements to an Output Manager. The Output Manager sends a "status" message when the request completes or fails or when an anomaly arises on the printer.

Draw Manager/Other Processes—The Draw Manager accepts lists of elements prefixed by explicit pixel parameters (density, scaling factor, etc.). It returns a single message containing a list of bit-map ("symbol") elements of the drawn result for each message it receives.

HUMAN INTERFACE—SERVICE

A Human Interface service is accessed by sending a request message to the closest (i.e. the "next") Human Interface manager, or directly to a specific Console Manager. This establishes a "connection" to an existing Human Interface resource or creates a new one. Subsequent requests must be made directly to the resource, using the connector returned from the initial request, until the connection is broken. The Human Interface manager is distributed and thus spans the entire virtual machine. Resources are associated with specific nodes.

A picture may be any size, often larger than any physical screen or window. A window may only be as large as the screen on which it appears. There may be any number of windows simultaneously displaying pictures on a single screen. Updating a picture which is mapped to a window causes the screen display to be updated automatically. Several windows may be mapped to the same picture concurrently—at different coordinates.

The input model provided by the Human Interface consists of two levels of "virtual devices" The lower level supports "position", "character", "action", and "function key" devices associated with a particular window. These are supported consistently regardless of the actual devices connected to the system.

An optional higher level consists of a "dialog service", which adds "icons", "menus", "prompts", "values", and "information boxes" to the repertoire of device-independent interaction. Input is usually event-driven (via messages) but may also be sampled or explicitly requested.

All dimensions are in terms of "virtual pixels" A virtual pixel is a unit of measurement which is symmetrical in both dimensions. It has no particular size. Its sole purpose is to define the spatial relationships between picture elements. Actual sizes are determined by the output device to which the picture is directed, if and when it is displayed. One virtual pixel may translate to any multiple, including fractions, of a real pixel.

Using the core Human Interface services generally involves: creating a picture (or accessing a predefined picture); creating a window on a particular screen and connecting the picture to it; updating the picture (drawing new elements, moving or erasing old ones, etc.) to reflect changes in the application (e.g. new data); if the application is interactive, repeatedly accepting input from the window and acting accordingly; and deleting the picture and/or window when done.

Creating a new resource is done with an appropriate "create" message, directed to the appropriate resource manager (i.e. the Human Interface manager or Console Manager). Numerous options are available when a resource, particularly a window, is created. For example, a typical application may want to be notified when a specific key is pressed. Pop-up and pull-down menus, and function keys, may also be defined for a window.

All input from the Human Interface is sent by means of the "click" message. The intent of this message is to allow the application program to be as independent of the external input as possible. Consequently, a "click" generated by a pop-up menu looks very much like that generated by pressing a function key or selecting an icon. Event-driven input is initiated by a user interacting with an external device, such as a keyboard or mouse. In this case, the "click" is sent asynchronously, and multiple events are queued.

A program may also explicitly request input, using a menu, prompt, etc., in which case the "click" is sent only when the request is satisfied. A third method of input, which doesn't directly involve the user, is to query the current state of a virtual input device (e.g., the current cursor position).

A "click" message is associated with a particular window (and by implication usually with a particular picture), or with a dialog "metaphor", thus reflecting the two levels of the input model.

Since the visual aspect of the Human Interface is separated from the application aspect, a later redesign of a window, menu, icon, etc. has little or no effect upon existing applications.

HUMAN INTERFACE—DETAILED DESCRIPTION

Connectors

In general, all interaction with a Human Interface resource (console, window, picture, or virtual terminal) must be through a connector to that resource. Connectors to consoles can only be obtained from the Human Interface manager. Connectors to the other resources are available through the Human Interface manager, or through the Console Manager in which the desired resource resides. Requests must specify the path-name of the resource as follows:

[<console_name>][/<screen_name>][/<window_or_picture_name>]

That is, the name of the console, optionally followed by a slash and the name of the screen, optionally followed by a slash and the name of a window, picture, or terminal. The console name may be omitted only if the message is sent directly to the desired console manager. If the screen name is omitted, the first screen configured on the given console is assumed. The window name must be specified if one of those resources is being connected.

Connection Requests

The "create" and "open" requests can be addressed to the "next" Human Interface context ("HI") or to a specific console connector or to the "next" context named "Console". If sent to "HI", a full path-name (the name parameter) must be given; otherwise, only the name of the desired resource is required (e.g., at a minimum, just the name of the window or picture).

If a picture manager process is created locally by an application, for private use, an "init" message—with the same contents as "create" or "open" must be sent directly to the picture process. The response will be "done" or "failed".

The following are the various Connection Requests and the types of information which may be associated with each:

CREATE is used to create a new picture resource, a new window resource, or a new virtual terminal resource.

When used to create a new picture resource, it may contain information about the resource type (i.e. a "picture"); the path-name of the picture; the size; the background color; the highlighting method; the maximum number of elements; the maximum element size; and the path-name of a library picture from which other elements may be copied.

When used to create a new window resource, it may contain information about the resource type (i.e. a "window"); the path-name of the window; the window's title; the window's position on the screen; the size of the window; the color, width, fill color between the outline and the pane, and the style of the main window outline; the color and width of the pane outline; a mapping of part of a picture into the window; a modification notation; a special character notation; various options; a "when" parameter requesting notification of various specified actions on/within the window; a title bar; a palette bar; vertical and horizontal scroll bars; a general use bar; and a corner box.

When used to create a new virtual terminal, it may contain information about the resource type (i.e. a "terminal"); the path-name of the terminal; the title of the terminal's window; various options; the terminal's position on the screen; the size of the terminal (i.e. number of lines and columns in the window); the maximum height and width of the virtual screen; the color the text inside the window; tab information; emulator process information; connector information to an existing window; window frame color; a list of menu items; and alternative format information.

OPEN is used to connect to a Human Interface service or to an existing Human Interface resource. When used to connect to a Human Interface service, it may contain information about the service type; and the name of the particular instance of the service. This resource must be sent to the Human Interface context.

When used to connect to an existing Human Interface resource, it may contain information about the path-name of the resource; the type of resource (e.g. picture, window, or terminal); and the name of the file (for pictures only) from which to load the picture. This request can be sent to a Human Interface manager or a console manager; alternatively the same message with message I.D. "init" specifying a file can be sent directly to a privately owned picture manager.

DELETE is used to remove an existing Human Interface resource from the system, and it may contain information specifying a connection to the resource; the type of resource; and whether, for a window, the corresponding picture is to be deleted at the same time.

CLOSE is used to break a connection to a Human Interface resource, and it may contain information specifying a connection to the resource; and the type of resource.

WHO? is used to request a list of signed-on users, and it may contain a user identification string.

QUERY is used to get the status of a service or resource, and it may contain information about the resource type; the name of the service or resource; a connector to a resource; and information concerning various options.

The following are the various Connection Responses and the types of information which may be associated with each:

CONNECT provides a connection to a Human Interface resource, and it contains information concerning the originator (i.e. the Human Interface or the console); the resource type; the original request message identifier; the name of the resource; and a connector to the resource.

USER contains the names of zero or more currently signed-on users and their locations, and it contains a connector to a console manager followed by the name of the user signed on at that console.

Console Requests

The main purpose of the console is to coordinate the activities of the windows, pictures, and dialog associated with it. Any of the CREATE, OPEN, DELETE, and CLOSE connection requests listed above, except those relating to the consoles, can be sent directly to a known console manager, rather than to the Human Interface manager (which always searches for the console by name). Subsequently, some characteristics of a window, such as its size, can be changed dynamically through the console manager. The current "user" of the console can be changed. And the console can be queried for its current status (or that of any of its resources).

The following are the various Console Requests and the types of information which may be associated with each:

USER is used to change the currently signed-on user, and it contains a user identification string.

CHANGE is used to change the size and other conditions of a window, and it may contain information about a connector to a window or a terminal; new height and width (in virtual pixels); increment to height and width; row and column position; various options; a connector to a new owner process; and whether the window should be the current active window on the screen.

CURSOR is used to move the screen cursor, and it contains position information as to row and column.

QUERY is used to get the current status of the console or one of its resources, and it contains information in the form of a connector to the resource; and various query options (e.g. list all screens, all pictures, or all windows).

BAND starts/stops the rubber-banding function and dragging function, and it contains information about the position of a point in the picture from which to start the operation; the end point of the figure which is to be dragged; the type of operation (e.g. line, rectangle, circle, or ellipse); the color; and the type of line (e.g. solid). In rubber-banding the drawn figure changes in size as the cursor is moved. In dragging the figure moves with the cursor.

The following are the various Console Responses and the types of information which may be associated with each:

STATUS describes the current state of a console, and it may contain information about a connector to the console; the originator; the name of the console; current cursor position; current metaphor size; scale of virtual pixels per centimeter, vertically and horizontally; number of colors supported; current user i.d. string; screen size and name; window connector and name; and picture connector, screen name, and window name.

Picture-Drawing

The picture is the fundamental building block in the Human Interface. It consists of a list of zero or more "picture elements", each of which is a device-independent abstraction of a displayable object (line, text, etc.). Each currently active picture is stored and maintained by a separate picture manager. "Drawing" a picture consists of sending picture manipulation messages to the picture manager.

A picture manager must first be initialized by a CREATE or OPEN request (or INIT, if the picture was created privately). CREATE sets the picture to empty, gives it a name, and defines the background. The OPEN request reads a predefined picture from a file and gives it a name. Either must be sent first before anything else is done. A subsequent OPEN reloads the picture from the file.

The basic request is to WRITE one or more elements. WRITE adds new elements to the end of the current list, thus

The attributes relating to the "type" field if designated "array" are as follows:

Bitmap=a two-dimensional array (in row and column order) of color numbers; each number either defines a color in which a pixel is to be drawn, or is zero (in which the pixel is drawn in the background color); the origin of the array corresponds to the starting location of the element

Alt=an alternate text string which can be displayed on non-bit-mapped devices in place of the array

A discrete element is used to plot distinct points on the screen, optionally with lines joining them. Each point is specified by its coordinates relative to the element's "box" An explicit element (usually a single-character text element or a symbol element) may be given as the mark to be drawn at each point. If not, an asterisk is used. The resulting figure cannot be filled.

The attributes relating to the "type" field if designated "discrete" are as follows:

Mark=a picture element which defines the "mark" to be drawn at each point; if not applicable, a null-length element (i.e., a single integer containing the value zero) must be given for this field

Style Pat Thick =type, pattern, and thickness of the line (see "line" element above)

Join="Y" or "N" (or null, which is equivalent to "N"); if "Y", lines will be drawn to connect the marks

Points=two or more pairs of coordinates; each point is relative to the upper left corner of the "box" defined in the header

A "macro" element is a composite, made up of the preceding primitive element types ("text", etc.) and/or other macro elements. It can sometimes be thought of as "bracketing" other elements. The coordinates of the contained elements are relative to the absolute coordinates of the macro element. The "length" field of the macro element includes its own header and the "macro" field, plus the sum of the lengths of all of the contained elements. The "text" macro is useful for mixing different fonts and styles in single "unit" (word, etc.) of text.

The attributes relating to the "type" field if designated "macro" are as follows:

Macro=describes the contents of the macro element; may be one of following:

"N"—normal (contained elements are complete)

"Y"—list: same as "N", but only one sub-element at a time can be displayed; the others will be marked "hidden", and only the displayed element will be sent in response to requests ("copy", etc.); the "highlight" request will cycle through the sub-elements in order

"T"—text: same as "N", but the "macro" field is immediately followed by a text "options" field, and a text "select" field; the macro "list" field may be followed by further text parameters (as specified in the options field)

List=any number of picture elements (referred to as sub-elements), formatted as described above; terminated by a null word

A "meta-element" is a pseudo-element generated by the picture manager and which describes the picture itself, whenever the picture is "saved" to a file. Subsequently, meta-elements read from a file are used to set up parameters pertinent to the picture, such as its size and background color. Meta-elements never appear in "write" messages issued by the picture manager (e.g. in response to a "read" request, or as an update to a window manager).

The format of the meta-element includes a length field, a type field, a meta-type field, and a value. The 16-bit length field always specifies a length of 36. The type field is like that for normal picture elements. The meta-element field contains one of the following types:

Name=the value consists of a string which names the picture

Size=the maximum row and column, and the maximum element number and size

Backgnd—the picture's background color

Highlt=the picture's highlighting

The format of the value field depends upon the meta-type.

Windowing

A window maps a particular subset (often called a "view") of a given picture onto a particular screen. Each window on a screen is a single resource which handles the "pane" in which the picture is displayed and up to four "frame bars".

With reference to FIG. 11, a frame bar is used to show ancillary information such as a title. Frame bars can be interactive, displaying the names of "pull-down" menus which, when selected, display a list of options or actions pertinent to the window. A palette bar is like a permanently open menu, with all choices constantly visible.

Scroll bars indicate the relative position of the window's view in the picture and also allow scrolling by means of selectable "scroll buttons" A "resize" box can be selected to expand or shrink the size of the window on the screen while a "close" box can be selected to get rid of the window. Selecting a "blow-up" box expands the window to full screen size; selecting it again reduces it to its original dimensions.

A corner box is available for displaying additional user information, if desired.

The window shown in FIG. 11 comprises a single pane, four frame bars, and a corner box. The rectangular element within each scroll bar indicates the relative position of the window in the picture to which it is mapped (i.e. about a third of the way down and a little to the right).

Performing an action (such as a "select") in any portion of the window will optionally send a "click" message to the owner of the window. For example, selecting an element inside the pane will send "click" with "action"="select" and "area"="inside", as well as the coordinates of the cursor (relative to the top left corner of the picture) and a copy of the element at that position.

Selecting the name of a menu, which may appear in any frame bar, causes the menu to pop-up. It is the response to the menu that is sent in the "click" message, not the selection of the menu bar item. Pop-up menus (activated by menu keys on the keyboard) and function keys can also be associated with a particular window.

All windows are created by sending a "create" request to a Console Manager. As described above, "create" is the most complex of the windowing messages, containing numerous options which specify the size and location of the window, which frame bars to display, what to do when certain actions are performed in the window, and so on.

The process which sent the request is known as the "owner" of the window, although this can be changed dynamically. The most recently opened window usually becomes the current "active" window, although this may be overridden or changed.

A subsequent "map" request is necessary to tell the window which picture to display (if not specified in the "create" request). "Map" can be re-issued any number of times.

Other requests define pop-up menus and soft-keys or change the contents of specific frame bars. A window is always opened on top of any other window(s) it overlaps. Depending upon the background specified for the relevant picture, underlying windows may or may not be visible.

The "delete" request unmaps the window and causes the window manager to exit. The owner of the window (if different from the sender of "delete") is sent a "status" message as a result.

The following are the various Windowing Requests and the types of information which may be associated with each:

MAP is used to map or re-map a picture to the window, and it may contain information specifying a connection to the desired picture; and the coordinates in the picture of the upper left corner of the "viewport", which will become [0,0] in the window's coordinate system.

UNMAP is used to disconnect a window from its picture, and it contains no parameters,

QUERY is used to get a window's status, and it contains no parameters.

[,] is used to start/ed a "batch", and the presence of a first symbol causes all updates to be postponed until a second symbol is received (batches may be nested up to 10 deep).

MENU defines a menu which will "pop-up" when a menu key is pressed, and it may contain information specifying which menu key will activate the menu; the name of the menu in the Human Interface library (if omitted, "list" must be given); and a name which is returned in the "click" message.

KEYS defines "pseudo-function" keys for the window, and it may contain information specifying the name of a menu in the Human Interface library; a list of key-names; and a name to be returned in the "click" message.

ADD, COPY, ERASE, REPLACE control elements in a frame bar, and they may contain information specifying the type of bar (e.g. title, palette, general, etc.); a list of picture elements for "add" and "replace" (omitted for "copy" and "erase"); and a tag identifying a particular element (not applicable to "add").

HIGHLIGHT, INVERT, HIDE, BLINK change attributes in a frame bar element, and they may contain information specifying a set/clear attribute; the type of bar; and a tag identifying a particular element in the bar.

The following are the various Windowing responses and the types of information which may be associated with each:

STATUS describes the current status of the window, and it may contain information specifying a connector to the window; specifying the originator (i.e. "window"); an original message identifier, if applicable; the subsystem; the name of the window; a connector to the window's console manager; the position of the window on the screen; the pane size and location; a connector to the picture currently mapped to the window; and the size and position of the view.

BAR represents a request to a "copy" request, and it may contain information specifying the type of bar (e.g. title, palette, general, corner box, etc.); and a list of picture elements.

CLICK describes a user-initiated event on or inside the window, and it may contain information specifying what event (e.g. inside a pane, frame bar, corner box, pop-up menu, function key, etc.); a connector to the window manager; a connector to the window's Console Manager; the name of the window; a menu or function-key name; a connector to the associated picture manager; a label from a menu or palette bar item or from a function key; the position of the cursor where the action occurred; the action performed by the user; a copy of the elements at the particular position; the first element's number; the first element's identifier; a copy of the character typed or a boundary indicator or the completion character; and other currently selected elements from all other windows, if any.

Virtual Terminal

In general, a virtual terminal window's behavior emulates that of a particular "real" terminal. If no particular emulation is requested, a simple "generic" terminal is provided.

The virtual terminal resource creates a picture of the given dimensions to represent the virtual "screen". The "screen" is strictly text-oriented and is organized as lines and characters, as reflected in messages. The virtual screen is displayed in a default window created by the terminal manager.

The following are the various Virtual Terminal requests and the types of information which may be associated with each:

WRITE sends the output to a terminal window, and it may contain information specifying a connector to the virtual terminal; the characters to be written; the data type; and the position on the virtual screen.

READ gets input from a terminal window, and it may contain information specifying a connector to the virtual terminal; an optional prompt string; a parameter to protect typed input (i.e. don't "echo"); continuous read (i.e. automatically re-issue the request at the end of every input line); the maximum number of characters to return; and the position on the virtual screen.

CANCEL terminates outstanding requests from processes, it contains no parameters.

SCROLL shifts a subset of lines up or down (inserts blank lines to fill a gap), and it may contain information specifying a starting and ending line number; and the number of lines to shift.

The following are the various Virtual Terminal responses and the types of information which may be associated with each:

STATUS describes the current state of the terminal, and it may contain information specifying a connector to the terminal; specifying the originator (i.e. the "terminal"); an original message identifier, if applicable; the name of the terminal; the height and width in characters; and the name of the emulator (if any).

WRITE is a response from a virtual terminal "read", and it may contain information specifying the name of the terminal; a connector to the terminal; specifying the originator (i.e. the "terminal"); the characters read, followed by a null character; the data type; and the character position within the terminal's "virtual screen"

Dialog Service

The dialog service provides representation-independent interaction with a user (as compared with device-independence, which is at a lower level). To a large extent programmers can ignore how prompts, error messages, etc. are

Preceding an item with "+" indicates that the item is currently "active" and causes a check mark to be displayed beside it whenever the menu is opened. Preceding an item with "–" indicates that the corresponding option is not currently available and cannot be selected.

An "arguments" string can be appended to the tag of an element in the menu. The string is passed "as is" to the application when the item is selected.

PROMPT

The greater part of a prompt picture comprises text which asks a question, often with some introductory preamble. One element, located anywhere in the picture, may represent a response area. This is generally a rectangular area into which a user can type the information requested by the prompt. This element must be tagged "RESP".

Two further elements, tagged "ENTER" and "CANCEL", display target text or symbols which are used to complete the prompt. When the "enter" element is selected by the user, the text typed in the response area is returned to the originator of the prompt.

If the "cancel" element is selected instead, the prompt is cancelled with a null response. The response element is optional. If omitted, the "enter" and "cancel" elements effectively correspond to "yes" or "no" responses. Typing a "carriage return" character will have the same effect as selecting "enter" The prompt is erased when any response is given, or by an explicit "cancel" request.

INFORMATION

An information picture comprises text (and possibly graphics) which describes something. One element, located anywhere in the picture, is usually tagged "DONE" When this element is selected, the information picture is erased from the display. If no such element is given, the process which requested the information to be displayed must send an explicit "cancel" request when it wants to get rid of it.

INPUT/OUTPUT DEVICE INDEPENDENCE

In the present invention all system interaction with the outside world is either through "virtual input" or "virtual output" devices. The system can accept any form of input or output device. The Human Interface is constructed using a well-defined set of "virtual devices" All Human Interface functions (windowing, picture—drawing, dialog management, etc.) use this set of devices exclusively.

These virtual input devices take the form of "keys" (typed textual input); "position" (screen coordinates); "actions" (Human Interface functions such as "open window", etc.); "functions" (user-defined actions); and "means" (pop-up lists of choices).

Virtual output devices produce device-independent output: text, lines, rectangles, polygons, circles, ellipses, discrete points, bit-mapped symbols, and bit-mapped arrays.

FIG. 12 shows how the console manager operates upon virtual input to generate virtual output. The lowest layer of HI software converts input from any "real" physical devices to the generic, virtual form, and it converts Human Interface output (in virtual form) to physical output.

FIG. 12 shows the central process of the Human Interface, the console manager 220, dealing with virtual input and producing virtual output. Virtual input passed through the virtual input manager 221 is processed directly by the console manager 220, while output is passed through two intermediate processes—(1) a picture manager 222, which manipulates device—independent graphical images, and (2) a window manager 224, which presents a subset (called a "view") of the overall picture to the virtual output manager 226.

Any number of physical devices can be connected to the Human Interface and can be removed or replaced dynamically, without disturbing the current state of the Human Interface or of any applications using the Human Interface. In other words, the Human Interface is independent of particular I/O devices, and the idiosyncracies of the devices are hidden from the Human Interface.

FIG. 13 represents a flowchart showing how virtual input is handled by the console manager. The virtual input may take any of several forms, such as a keystroke, cursor position, action, function key, menu, etc.

For example, regarding the operations beneath block 231, if the virtual input to the console manager is a keystroke, then the console manager checks to see whether the cursor is inside a window. If so, it checks to see whether it originated from a virtual terminal, and if not it checks to see whether an edit operation is taking place. If not, it updates the picture.

Regarding the operations beneath block 232, if the virtual input represents a cursor position, then the console manager checks to see whether the auto-highlight option has been enabled. If yes, it checks to see whether the cursor is on an element. If so it highlights that element.

Regarding the operations beneath block 233, the console manager uses any of the indicated actions to update a picture, update a window, or initiate dialog, as appropriate.

Regarding the operations beneath block 234, if the virtual input is from a function key, the console manager notifies the dialog manager.

Regarding the operations beneath block 235, if the virtual input represents a menu choice, the console manager checks to see whether the cursor is in a window. If not, it determines that it is on a user metaphor; if so, it requests a menu from the window. If the menu is defined, it notifies the owner of the window (or metaphor), activates a pop-up menu, gets a response, and sends the response to the window owner.

FIG. 14 represents a flowchart showing how virtual output is handled by the picture manager. The picture manager 240 accepts virtual output from the console manager and then, depending upon the type of operation, performs the requested function. For example, if the operation is a replace operation, the picture manager 240 replaces the old output with the new and sends the change(s) to the window manager. The window manager sends the change to the output manager, which in turn sends it to the real device.

DESCRIPTION OF SOURCE CODE LISTING

Program Listings A and B contain a "C" language implementation of the above-described concepts relating to input/output device independence. The following chart indicates where the relevant portions of the listing may be found.

Function	
	Lines Numbers in Program Listing A
Main-line; initialization; accept input	190–222
Determine type of input	486–521
Virtual key	523–631
Virtual position	633–661
Virtual action	663–702, 763–1200
Virtual function	704–723
Virtual menu	725–761
	Lines Numbers in Program Listing B
Main-line; initialization; start processing	125–141
Accept requests (virtual output); check for changes	161–203
Determine type of request	239–310

What is claimed is:

1. A virtual input interface in a data processing system, said interface comprising:

means for accepting input from at least one physical device and for converting said physical device input into virtual input, said means comprising a virtual input manager process responsive to said at least one physical input device for generating a picture, said picture comprising one or more picture elements, each picture element comprising a plurality of device-independent data structures in a predetermined, standard data format, at least one of said data structures comprising a plurality of different data fields each containing information describing said picture element; and

means responsive to said virtual input for performing processing operations upon said virtual input, said means comprising a console manager process for performing processing operations on said one or more picture elements.

2. The virtual input interface as recited in claim 1, wherein said input accepting means accepts input in the form of keystrokes.

3. The virtual input interface as recited in claim 1, wherein said input accepting means accepts input in the form of cursor position.

4. The virtual input interface as recited in claim 1, wherein said input accepting means accepts input in the form of system-defined actions.

5. The virtual input interface as recited in claim 1, wherein said input accepting means accepts input in the form of user-defined functions.

6. The virtual input interface as recited in claim 1, wherein said input accepting means accepts input in the form of menu selections.

7. The virtual input interface as recited in claim 1, wherein said at least one physical device can be removed from said system without affecting the operation of the remainder of said system.

8. The virtual input interface as recited in claim 1, wherein at least one additional physical device can be added to said system without affecting the operation of the remainder of said system.

9. A virtual output interface in a data processing system, said interface comprising:

a source of virtual input, said virtual input comprising one or more picture elements, each picture element comprising a plurality of device-independent data structures in a predetermined, standard data format, at least one of said data structures comprising a plurality of different data fields each containing information describing said picture element;

means for performing processing operations on said virtual input and for generating virtual output;

means for accepting said virtual output; and

means for converting said virtual output into at least one physical output suitable for use by at least one physical output device.

10. The virtual output interface as recited in claim 9, wherein said virtual input comprises a plurality of related picture elements and wherein said virtual output accepting means comprises a picture manager process for controlling said plurality of related picture elements.

11. The virtual output interface as recited in claim 10 and further comprising a display device, wherein said virtual output accepting means further comprises a window manager process for controlling the display of said plurality of related picture elements on said display device.

12. The virtual output interface as recited in claim 9, wherein said virtual output converting means comprises a virtual output manager process responsive to said one or more processed picture elements for coupling said one or more processed picture elements to said at least one physical output device.

13. The virtual output interface as recited in claim 9, wherein said at least one physical device can be removed from said system without affecting the operation of the remainder of said system.

14. The virtual output interface as recited in claim 9, wherein at least one additional physical device can be added to said system without affecting the operation of the remainder of said system.

15. In a data processing system, an interface between processes and data in said system and physical input and output devices coupled to said system, said interface comprising:

means responsive to one of said physical input devices for generating a picture, said picture comprising one or more picture elements, each picture element comprising a plurality of device-independent data structures in a predetermined, standard data format, at least one of said data structures comprising a plurality of different data fields each containing information describing said picture element;

means for performing processing operations on said one or more picture elements; and

means responsive to said one or more processed picture elements for coupling said one or more processed picture elements to one of said physical output devices.

16. The data processing system as recited in claim 15, wherein said one or more picture elements define a graphical object and at least one attribute thereof.

17. The data processing system as recited in claim 16, wherein one of said data fields describes the length of the associated picture element.

18. The data processing system as recited in claim 16, wherein one of said data fields identifies the particular type of the associated picture element.

19. The data processing system as recited in claim 16, wherein one of said data fields describes the position of the associated picture element relative to row and column coordinates on a picture of which said picture element forms a part.

20. The data processing system as recited in claim 16, wherein one of said data fields describes the size of the associated picture element.

21. The data processing system as recited in claim 16, wherein one of said data fields describes the color of the associated picture element.

22. The data processing system as recited in claim 15, wherein said means responsive to one of said physical input devices comprises a virtual input manager process.

23. The data processing system as recited in claim 15, wherein said means responsive to said one or more processed picture elements comprises a virtual output manager process.

* * * * *

EXHIBIT

34

US005502839A

United States Patent [19]

Kolnick

[11] **Patent Number: 5,502,839**

[45] **Date of Patent: Mar. 26, 1996**

[54] **OBJECT-ORIENTED SOFTWARE ARCHITECTURE SUPPORTING INPUT/OUTPUT DEVICE INDEPENDENCE**

[75] Inventor: **Frank C. Kolnick**, Willowdale, Canada

[73] Assignee: **Motorola, Inc.**, Schaumburg, Ill.

[21] Appl. No.: **361,738**

[22] Filed: **Jun. 2, 1989**

Related U.S. Application Data

[63] Continuation of Ser. No. 619, Jan. 5, 1987, abandoned.

[51] **Int. Cl.**[6] **G06F 13/00**

[52] **U.S. Cl.** **395/800**; 364/228.2; 364/237.9; 364/239.9; 364/280; 364/284.2; 364/DIG. 1

[58] **Field of Search** .... 364/200 MS File, 364/900 MS File; 395/500

[56] **References Cited**

U.S. PATENT DOCUMENTS

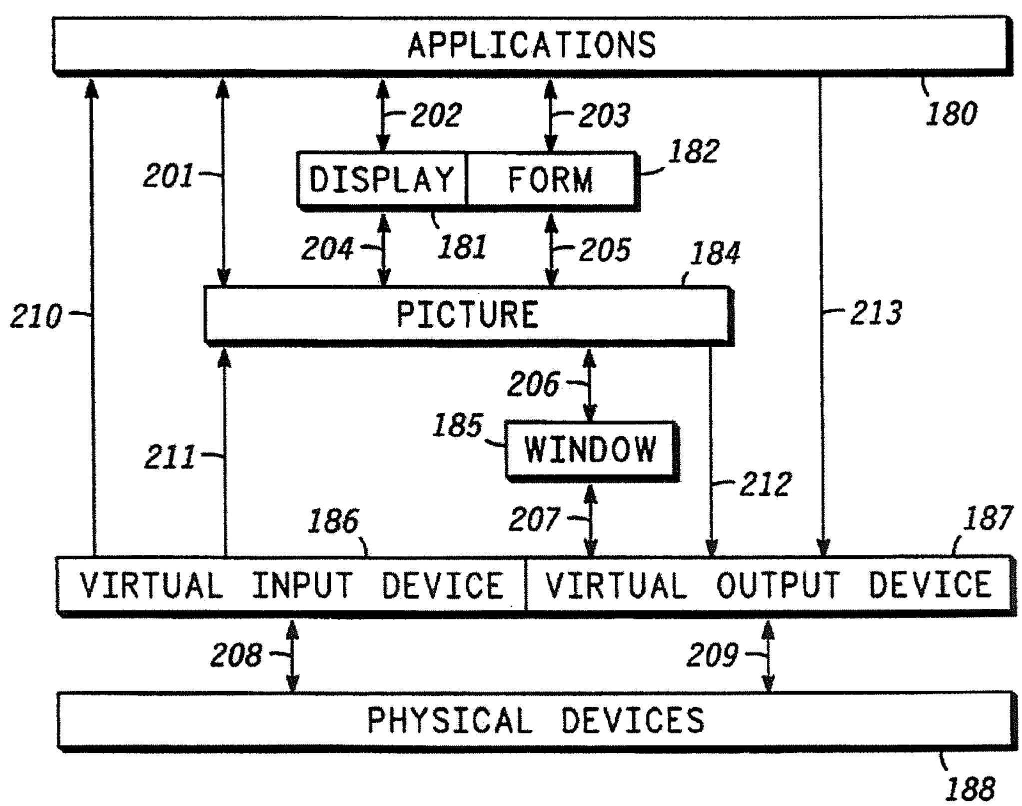
3,930,232	12/1975	Wallach et al.	395/500
4,241,341	12/1980	Thorson	340/747
4,454,593	6/1984	Fleming	364/900
4,485,439	11/1984	Rothstein	395/500
4,547,628	10/1985	Tamura et al.	340/747
4,555,775	11/1985	Pike	364/900
4,559,614	12/1985	Peek et al.	364/900
4,642,790	2/1987	Minshull et al.	364/900
4,754,395	6/1988	Weisshaar et al.	364/200
4,800,523	1/1989	Gerety et al.	395/500
4,858,114	8/1989	Heath et al.	395/500
5,063,494	11/1991	Davidowski et al.	395/800

Primary Examiner—Kevin J. Teska
Assistant Examiner—Ayni Mohamed
Attorney, Agent, or Firm—Walter W. Nielsen; Harold C. McGurk; S. Kevin Pickens

[57] **ABSTRACT**

An object-oriented software architecture interacts with "real" input/output devices exclusively through "virtual" input/output devices. Since all human interface with the operating system is performed through such virtual devices, the system can accept any form of real input or output devices. The lowest level of the operating system converts input from any physical device to virtual form and converts virtual output into suitable physical output. Any number of physical devices can be connected to, removed from, or replaced in the system without disrupting the system.

23 Claims, 9 Drawing Sheets



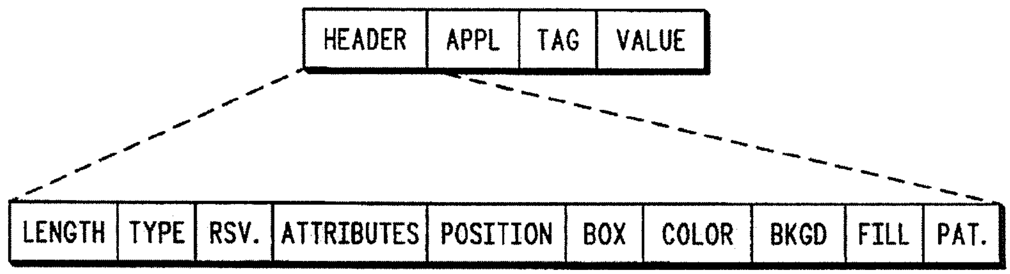
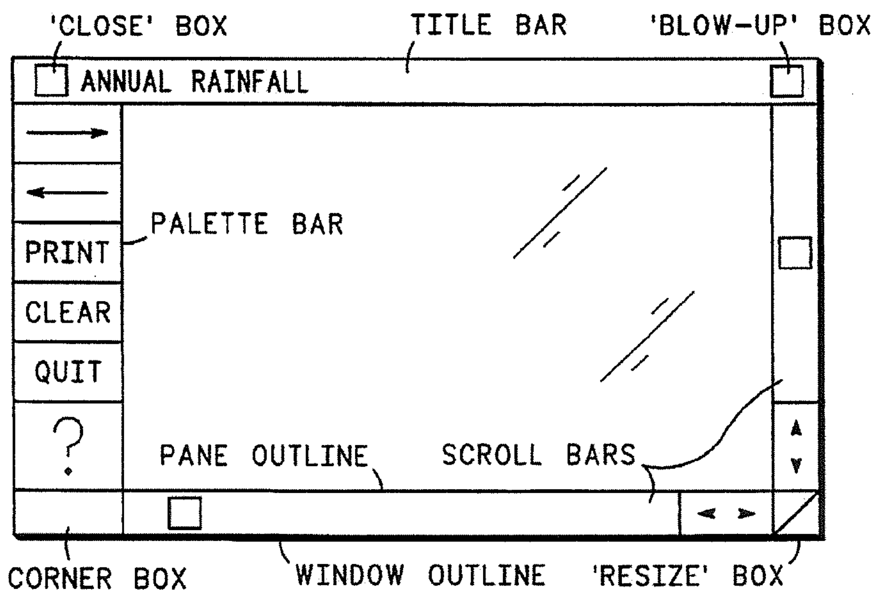

HEADER
APPL
TAG
VALUE
LENGTH
TYPE
RSV.
ATTRIBUTES
POSITION
BOX
COLOR
BKGD
FILL
PAT.

FIG. 10

FIG. 11


'CLOSE' BOX
TITLE BAR
'BLOW-UP' BOX
ANNUAL RAINFALL
PALETTE BAR
PRINT
CLEAR
QUIT
?
PANE OUTLINE
SCROLL BARS
CORNER BOX
WINDOW OUTLINE
'RESIZE' BOX

start-up. These processes then start up their respective subsystems. A node configuration service on each node sends configuration messages to each subsystem when it is being initialized, informing it of the devices it owns. Thereafter, similar messages are sent whenever a new device is added to the node or a device fails or is removed from the node.

Thus there is no well-defined meaning for "system up" or "system down"—as long as any node is active, the system as a whole may be considered to be "up". Nodes can be shut down or started up dynamically without affecting other nodes on the network. The same principle applies, in a limited sense, to peripherals. Devices which can identify themselves with regard to type, model number, etc. can be added or removed without operator intervention.

FIG. **6** shows the standard format of a message in a distributed data processing system of the type incorporating the present invention. The message format comprises a message i.d. portion **150**; one or more "triples" **151, 153,** and **155**; and an end-of-message portion **160**. Each "triple" comprises a group of three fields, such as fields **156–158**.

The first field **156** of "triple" **151**, designated the PCRT field, represents the name of the process to be created. The second field **157** of "triple" **151** gives the size of the data field. The third field **158** is the data field.

The first field **159** of "triple" **153**, designated the PNTF field, represents the name of the process to notify when the process specified in the PCRT field has been created.

A message can have any number of "triples", and there can be multiple "triples" in the same message containing PCRT and PNTF fields, since several processes may have to be created (i.e. forming a context, as described hereinabove) for the same resource.

As presently implemented, portion **150** is 16 bytes in length, field **156** is 4 bytes, field **157** is 4 bytes, field **158** is variable in length, and EOM portion **160** is 4 bytes.

HUMAN INTERFACE—GENERAL

The Human Interface of the present invention provides a set of tools with which an end user can construct a package specific to his applications requirements. Such a package is referred to as a "metaphor", since it reflects the user's particular view of the system. Multiple metaphors can be supported concurrently. One representative metaphor is, for example, a software development environment.

The purpose of the Human Interface is to allow consistent, integrated access to the data and functions available in the system. Since users' perceptions of the system are based largely on the way they interact with it, it is important to provide an interface with which they feel comfortable. The Human Interface allows a systems designer to create a model consisting of objects that are familiar to the end user and a set of actions that can be applied to them.

The fundamental concept of the Human Interface is that of the "picture". All visually-oriented information, regardless of interpretation, is represented by pictures. A picture (such as a diagram, report, menu, icon, etc.) is defined in a device-independent format which is recognized and manipulated by all programs in the Human Interface and all programs using the Human Interface. It consists of "picture elements", such as "line", "arc", and "text", which can be stored compactly and transferred efficiently between processes. All elements have common attributes like color and fill pattern. Most also have type-specific attributes, such as typeface and style for text. Pictures are drawn in a large "world" co-ordinate system composed of "virtual pixels".

Because all data is in the form of pictures, segments of data can be freely copied between applications, e.g., from a live display to a word processor. No intermediate format or conversion is required. One consequence of this is that the end user or original equipment manufacturer (OEM) has complete flexibility in defining the formats of windows, menus, icons, error messages, help pages, etc. All such pictures are stored in a library rather than being built into the software and so are changeable at any time without reprogramming. A comprehensive editor is available to define and modify pictures on-line.

All interaction with the user's environment is through either "virtual input" or "virtual output" devices. A virtual input device accepts keyboards, mice, light pens, analog dials, pushbuttons, etc. and translates them into text, cursor-positioning, action, dial, switch, and number messages. All physical input devices must map into this set of standard messages. Only one process, an input manager for the specific device, is responsible for performing the translation. Other processes can then deal with the input without being dependent on its source.

Similarly, a virtual output manager translates standard output messages to the physical representation appropriate to a specific device (screen, printer, plotter, etc.) A picture drawn on any terminal or by a process can be displayed or printed on any device, subject to the physical limitations of that device.

With reference to FIG. **7**, two "pictures" are illustrated picture A (**170**) and picture B (**174**).

The concept of a "view" is used to map a particular rectangular area of a picture to a particular device. In FIG. **7**, picture A is illustrated as containing at least one view **171**, and picture B contains at least one view **175**. Views can be used, for example, to partition a screen for multiple applications or to extract page-sized subsets of a picture for printing.

If the view appears on a screen it is contained in a "window". With reference again to FIG. **7**, view **171** of picture A is mapped to screen **176** as window **177**, and view **175** of picture B is mapped as window **178**.

The Human Interface allows the user to dynamically change the size of the window, move the window around on the screen, and move the picture under the window to view different parts of it (i.e., scroll in any direction). If a picture which is mapped to one or more windows changes, all affected views of that picture on all screens are automatically updated. There is no logical limit to the number or sizes of windows on a particular screen. Since the system is distributed, it's natural for pictures and windows to be on different nodes. For example, several alarm displays can share a single, common picture.

The primary mechanism for interacting with the Human Interface is to move the cursor to the desired object and "select" it by pressing a key or button. An action may be performed automatically upon selection or by further interaction, often using menus. For example, selecting an icon usually activates the corresponding application immediately. Selecting a piece of text is often followed by selection of a command such as "cut" or "underline" Actions can be dynamically mapped to function keys on a keyboard so that pressing a key is equivalent to selecting an icon or a menu item. A given set of cursors (the cursor changes as it moves from one application picture to another), windows, menus, icons, and function keys define a "metaphor".

The Human Interface builds on the above concepts to provide a set of distributed services. These include electronic mail, which allows two or more users at different terminals to communicate with each other in real time or to queue files for later delivery, and a forms manager for data entry. A subclass of windows called "virtual terminals" provides emulation of standard commercially available terminals.

FIG. 8 shows the different levels of the Human Interface and data flow through them. Arrows **201–209** indicate the most common paths, while arrows **210–213** indicate additional paths. The interface can be configured to leave out unneeded layers for customized applications. The philosophy behind the Human Interface design dictates one process per object. That is, a process is created for each active window, picture, input or output device, etc. As a result, the processes are simplified and can be distributed across nodes almost arbitrarily.

MULTIPLE INDEPENDENT PICTURES AND WINDOWS

A picture is not associated with any particular device, and it is of virtually unlimited size. A "window" is used to extract a specified rectangular area—called a "view"—of picture information from a picture and pass this data to a virtual output manager.

The pictures are completely independent of each other. That is, none is aware of the existence of any other, and any picture can be updated without reference to, and without affect upon, any other. The same is true of windows.

Thus the visual entity seen on the screen is really represented by two objects: a window (distinguished by its frame title, scroll bars, etc.), and a picture, which is (partially) visible within the boundaries of the window's frame.

As a consequence of this autonomy, multiple pictures can be updated simultaneously, and windows can be moved around on the screen and their sizes changed without the involvement of other windows and/or pictures.

Also, such operations are done without the involvement of the application which is updating the window. For example, if the size of a window is increased to look at a larger area of the picture, this is handled completely within the human interface.

HUMAN INTERFACE—PRIMARY FEATURES

The purpose of the Human Interface is to transform machine-readable data into human-readable data and vice versa. In so doing the Human Interface provides a number of key services which have been integrated to allow users to interact with the system in a natural and consistent manner. These features will now be discussed.

Device Independence—The Human Interface treats all devices (screens, printers, etc.) as "virtual devices". None of the text, graphics, etc. in the system are tied to any particular hardware configuration. As a result such representations can be entered from any "input" device and displayed on any "output" device without modification. The details of particular hardware idiosyncracies are hidden in low-level device managers, all of which have the same interface to the Human Interface software.

Picture Drawing—The Human Interface can draw "pictures" composed of any number of geometric elements, such as lines, circles, rectangles, etc., as well as any arbitrary shape defined by the user. Each element can have its own color and line thickness. In addition, closed figures may be filled in with a particular shading pattern in any given color. A picture can be of almost any size. All output from the Human Interface to a user is via pictures, and all input from a user to the Human Interface is stored as pictures, so that there is only one representation of data within the Human Interface.

Text can be freely intermixed with graphical images, so that the user need only learn one "editor" to do his job. Consequently it is not necessary to switch between editors or "cut and paste" between pictures. Text characters can be selected from a large predefined character set, which includes mathematical and Greek symbols, among others, and can be typed in a wide variety of fonts, colors, sizes, and styles (e.g. bold, italic, or underlined). It is also possible for a user to define his own symbols and add them to the character set.

Windowing—The Human Interface allows the user to partition a screen into as many "sub-screens" or "windows" as required to view the information he desires. The Human Interface places no restrictions on the contents of such windows, and all windows can be simultaneously updated in real time with data from any number of concurrently executing programs. Any picture can be displayed, created, or modified ("edited") in any window. Also any window can be expanded or contracted, or it can be moved to a new location on the screen at any time.

If the current picture is larger than the current window, the window can be scrolled over the picture, usually in increments of a "line" or a "page". It is also possible to temporarily expand or contract the visible portion of the picture ("zoom in" or "zoom out") without changing the window's dimensions and without changing the actual picture.

Dialog Management—The Human Interface is independent of any particular language or visual representation. That is, there are no built-in titles, menus, error messages, help text, icons, etc. for interacting with the system. All such information is stored as pictures which can be modified to suit the end user's requirements, either prior to or after installation. The user can modify the supplied dialog with his own at any time.

Data Entry—The Human Interface provides a generalized interface between the user and any program (such as a data base manager) which requires data from the user. The service is called "forms management", because a given data structure is displayed as a fill-in-the-blanks type of "form" consisting of numerous modifiable fields with descriptive labels. The Human Interface form is interactive, so that data can be verified as it is entered, and the system can assist the user by displaying explanatory text when appropriate (on demand or as a result of an error).

Communication Between Users—The Human Interface permits two or more users to "converse" with each other in real time or to send "mail" to each other. Conversation is performed through a window on each of the user's screens. Mail is sent by creating a picture (text and/or diagrams) and specifying a destination. The destination may be one particular user, a group of users, or all users in the system (i.e. a "broadcast"). Transmission may be immediate or delayed until a given date and time or until the given user(s) sign onto the system. When mail arrives at the destination, the receiving user is informed and may then read, save, print, or erase the picture.

Event Management—The Human Interface can record any arbitrary event for future reference. The Human Interface defines a simple, yet flexible grammar for forming "sentences" which describe events and which the Human Interface can use to parse in order to manipulate events for specific requests. For example, events can be dynamically displayed on a screen by time and/or priority, or they can be scanned for a particular "subject" or "object" or any other attribute. Each event can be time-stamped by the sender; if not, it is automatically time-stamped upon receipt.

The Human Interface records all of its own actions, such as printing a report or signing-on a user, and it provides this service to any applications program. In addition, the Human Interface can be requested to trigger any given action upon the occurrence of any given event, thus providing a kind of closed-loop control service to applications.

Modularity—The Human Interface comprises a number of separate software components which can be replicated and distributed throughout the hardware configuration to achieve optimal performance. For example, each time a new "console" (for example, keyboard plus screen) is connected to the system, a new "Console" component is created to manage it. There is no logical limit to the number of consoles that the Human Interface can handle. In general the relevant software component is located close to the hardware or other resources on which it most depends.

HUMAN INTERFACE—BASIC COMPONENTS

The Human interface comprises the following basic components:

Console Manager—It is the central component of a Console context and consequently is the only manager which knows all about its particular "console" It is therefore aware of all screens and keyboards, all windows, and all pictures. Its primary responsibility is to coordinate the activities of the context. This consists of starting up the console (initializing the device managers, etc.) creating and destroying pictures, and allocating and controlling windows for processes in the Human Interface and elsewhere. Thus all access to a console must be indirect, through the relevant Console Manager.

The Console Manager also implements the first level of Human Interface interaction, via menus, prompts, etc., so that applications processes don't have to. Rather than using built-in text and icons, it depends upon the Dialog Manager to provide it with the visible features of the system. Thus all cultural and user idiosyncracies (such as language) are hidden from the rest of the Human Interface.

A Console Manager knows about the following processes: the Output Manager(s) in its context, the Input Manager in its context, the Window Managers in its context, the Picture Managers in its context, and the Dialog Manager in its context. The following processes know about the Console Manager: any one that wants to.

When a Console Manager is started, it waits for the basic processes needed to communicate with the user to start up and "sign on". It this is successful, it is ready to talk to users and other processes (i.e., accept messages from the Input Manager and other processes). All other permanent processes in the context (Dialog, etc.) are assumed to be activated by the system start-up procedure. The "In" and "Cursor" processes (see "Input Manager" and "Output Manager" below) are created by the Console Manager at this time.

The Console Manager generally clears the entire screen and displays appropriate status text during the course of the start-up (by sending picture elements directly to its Output Manager(s)). If any part of the start-up fails, it displays appropriate "error" text and possibly waits for corrective action from a user.

The Console Manager views the screen as being composed of blank (unused) space, windows, and icons. Whenever an input character is received, the Console Manager determines how to handle it depending upon the location of the cursor and the type of input, as follows:

A. Requests to create or eliminate a window are handled within the Console Manager. A window may be opened anywhere on the screen, even on top of another window. A new Picture Manager and possibly a Window Manager may be created as a result, and one or more new messages may be generated and sent to them, or the manager(s) may be told to quit.

B. Icons can only be selected, then moved or opened. The Console Manager handles selection and movement directly. It sends notification of an "open" to the Dialog Manager, which sends a notification to the application process associated with the icon and possibly opens a default window for it.

C. For window-dependent actions, if the cursor is outside all windows, the input is illegal, and the Console Manager informs the user; otherwise the input is accepted. Request which affect the window itself (such as "scroll" or "zoom") are handled directly by the Console Manager. A "select" request is pre-checked, the relevant picture elements are selected (by sending a message to the relevant Picture Manager), and the message is passed on to the process currently responsible for the window. All other inputs are passed directly to the responsible process without being pre-checked.

If the cursor is on a window's frame, the only valid actions are to move, close, or change the dimensions of the window, or select an object in the frame (such as a menu or a scroll bar). These are handled directly by the Console Manager.

D. Requests for Human interface services not in the Console context are treated as errors.

A new window is opened by creating a new Window Manager process and telling it its dimensions and the location of its upper left corner on the screen. It must also be given the PID of a Picture Manager and the coordinates of the part of the picture it is to display, along with the dimensions of a "clipping polygon", if that information is available. (It is not possible to create a window without a picture.) The type and contents of the window frame are also specified. Any of these parameters may be changed at any time.

A new instance of a picture is created by creating a new Picture Manager process with the appropriate name and, optionally, telling it the name of a "file" from which to get its picture elements. If a file is not provided, an "empty" picture is created, with the expectation that picture-drawing requests will fill it in.

Menus, prompts, help messages, error text, and icons are simply predefined pictures (provided through the Dialog Manager) which the Console Manager uses to interact with users. They can therefore be created and edited to meet the requirements of any particular system the same way any picture can be created and edited. Menus and help text are usually displayed on request, although they may sometimes be a result of another operation.

Prompts are displayed when the system needs information from the user. Error text is displayed whenever the user tries to do something that is illegal or when the system is having problems of its own (e.g. "printer out of paper"). Icons are displayed by the Console Manager automatically when a specific frame of reference is requested by the user. The Console Manager may also display informational messages (such as "console starting up") which are automatically erased when the associated action is finished.

Picture Manager—It is created when a picture is built, and it exits when the picture is no longer required. There is one Picture Manager per picture. The Picture Manager constructs a device-independent representation of a picture using a small set of elemental "picture elements" and controls modification and retrieval of the elements.

A Picture Manager knows about the following processes: the process which created it, and the Draw Manager. The following processes know about the Picture Manager: the Console Manager in the same context, and Window Managers in the same context.

A Picture Manager is created to handle exactly one picture, and it need only be created when that picture is being accessed. It can be told to quit at any time, deleting its representation of the picture. Some other process must copy the picture to a file if it needs to be saved.

When a Picture Manager first starts up, its internal picture is empty. It must receive a "load file" request, or a series of "draw" requests, before a picture is actually available. Until that is done any requests which refer to specific elements or locations in the picture will receive an appropriate "not found" status message.

A picture is logically composed of device-independent "elements", such as text, line, arc, and symbol. In general, there is a small number of such elements. Each element consists of a common header, which includes the element's position in the picture's coordinate system, its color, size, etc., and a "value" which is unique to the element's type (e.g. a character string, etc.). The header also specifies how the element combines with other elements in the picture (overlays them, merges with them, etc.). A special element type called "null" is also supported to facilitate the removal of picture elements from pictures or other similar large lists without forcing time-consuming compaction procedures. Any element can therefore be redefined to "null", indicating that it should be ignored for all future processing.

The "null" color (zero) is treated as transparent when used in either the foreground or the background. Specifically, if the foreground color is null, the element itself is not drawn, but it may still be filled in. If the background color is null, the element is not filled in. If the shading pattern is null, and the color is not null, the background fill is solid.

A picture is represented in an internal format which may be different from the external representation of picture elements and which is, in any case, hidden from other processes. This representation is designed to optimize retrieval of picture elements, with a secondary emphasis on adding new elements and modifying or erasing old ones. The order in which the elements were originally drawn is preserved (unless explicit "order" requests have been received to re-arrange them).

Requests to "animate" an element result in the creation of a separate, local "animate" process which performs the necessary transformations and sends the appropriate requests (usually "draw" or "erase") back to the Picture Manager periodically.

A Picture Manager processes incoming requests one at at time, as it receives them. Each message can change the state of the picture for later requests. The Picture Manager supports numerous operations, including the following: "draw" new elements; "modify", "overwrite", or "erase" existing elements; "copy" or "move" elements to another location in the same picture or to any other given process; "group" elements together into one (or "ungroup" them); "scale" them (i.e. expand, stretch, or shrink them); and "rotate" them. It can also be asked to "notify" a particular process if any elements within a given rectangular area (the "viewport") are changed and to determined whether a given location coincides (or come close to) any element in the picture. Any response to a request (e.g., multiple picture elements) is sent in a single message.

When an element is sent as the result of an outstanding "notify" request, all elements which overlap it (and all elements which overlap those elements) are sent as well. These are sent together in one message. The background is displayed by generating a "rectangle" element of the same size as the current viewport with a null foreground color and the appropriate background pattern and color. This element is always the lowest level in the picture; i.e., it is sent before all others. All erasure of elements from a display is accomplished by "draw" requests which redisplay the background and/or elements in the picture, overwriting the "erased" elements. There is no explicit "erase" request to a window (or output) manager.

Input Manager—There is one Input Manager per set of "logical input devices" (such as keyboards, mice, light pens, etc.) connected to the system. The Input Manager handles input interrupts and passes them to the console manager. Cursor movement inputs may also be sent to a designated output manager.

The Input Manager knows about the following processes: the process which initialized it, and possibly one particular Output Manager in the same context. The following process knows about the Input Manager: the Console Manager in the same context.

An Input Manager is created (automatically, at system start-up) for each set of "logical input devices" in the system, thus implementing a single "virtual keyboard" There can only be one such set, and therefore one Input Manager, per Console context. The software (message) interface to each manager is identical, although their internal behavior is dependent upon the physical device(s) to which they communicate. All input devices interrupt service routines (including mouse, digitizing pad, etc.) are contained in Input Managers and hidden from other processes. When ready, each Input Manager must send an "I'm here" message to the closest process named "Console".

An Input Manager must be explicitly initialized and told to proceed before it can begin to process input interrupts. Both of these are performed using appropriate messages. Whichever process initializes the manager becomes tightly coupled to it, i.e., they can exchange messages via PID's rather than by name. The Input Manager will send all inputs to this process (usually the Console Manager). This coupling cannot be changed dynamically; the manager would have to be re-initialized, Between the "initialize" and the "proceed" an Input Manager may be sent one or more "set" requests to define its behavior. It does not need to be able to interpret the meaning of any input beyond distinguishing cursor from non-cursor. Device-independent parameters (such as pixel size and density) are not down-loaded but rather are assumed to be built into the software, some part of which, in general, must be unique to each type of Input Manager.

An Input Manager can be dynamically "linked" to a particular Output Manager, if desired. If so, all cursor control input (or any other given subset of the character set) will be sent to that manager, in addition to the initializing process, as it is received. This assignment can be changed or eters (density, scaling factor, etc.). It returns a single message containing a list of bit-map ("symbol") elements of the drawn result for each message it receives.

HUMAN INTERFACE—SERVICE

A Human Interface service is accessed by sending a request message to the closest (i.e. the "next") Human Interface manager, or directly to a specific Console Manager. This establishes a "connection" to an existing Human Interface resource or creates a new one. Subsequent requests must be made directly to the resource, using the connector returned from the initial request, until the connection is broken. The Human Interface manager is distributed and thus spans the entire virtual machine. Resources are associated with specific nodes.

A picture may be any size, often larger than any physical screen or window. A window may only be as large as the screen on which it appears. There may be any number of windows simultaneously displaying pictures on a single screen. Updating a picture which is mapped to a window causes the screen display to be updated automatically. Several windows may be mapped to the same picture concurrently—at different coordinates.

The input model provided by the Human Interface consists of two levels of "virtual devices" The lower level supports "position", "character", "action", and "function key" devices associated with a particular window. These are supported consistently regardless of the actual devices connected to the system.

An optional higher level consists of a "dialog service", which adds "icons", "menus", "prompts", "values", and "information boxes" to the repertoire of device-independent interaction. Input is usually event-driven (via messages) but may also be sampled or explicitly requested.

All dimensions are in terms of "virtual pixels" A virtual pixel is a unit of measurement which is symmetrical in both dimensions. It has no particular size. Its sole purpose is to define the spatial relationships between picture elements. Actual sizes are determined by the output device to which the picture is directed, if and when it is displayed. One virtual pixel may translate to any multiple, including fractions, of a real pixel.

Using the core Human Interface services generally involves: creating a picture (or accessing a predefined picture); creating a window on a particular screen and connecting the picture to it; updating the picture (drawing new elements, moving or erasing old ones, etc.) to reflect changes in the application (e.g. new data); if the application is interactive, repeatedly accepting input from the window and acting accordingly; and deleting the picture and/or window when done.

Creating a new resource is done with an appropriate "create" message, directed to the appropriate resource manager (i.e. the Human Interface manager or Console Manager). Numerous options are available when a resource, particularly a window, is created. For example, a typical application may want to be notified when a specific key is pressed. Pop-up and pull-down menus, and function keys, may also be defined for a window.

All input from the Human Interface is sent by means of the "click" message. The intent of this message is to allow the application program to be as independent of the external input as possible. Consequently, a "click" generated by a pop-up menu looks very much like that generated by pressing a function key or selecting an icon. Event-driven input is initiated by a user interacting with an external device, such as a keyboard or mouse. In this case, the "click" is sent asynchronously, and multiple events are queued.

A program may also explicitly request input, using a menu, prompt, etc., in which case the "click" is sent only when the request is satisfied. A third method of input, which doesn't directly involve the user, is to query the current state of a virtual input device (e.g., the current cursor position).

A "click" message is associated with a particular window (and by implication usually with a particular picture), or with a dialog "metaphor", thus reflecting the two levels of the input model.

Since the visual aspect of the Human Interface is separated from the application aspect, a later redesign of a window, menu, icon, etc. has little or no effect upon existing applications.

HUMAN INTERFACE—DETAILED DESCRIPTION

Connectors

In general, all interaction with a Human Interface resource (console, window, picture, or virtual terminal) must be through a connector to that resource. Connectors to consoles can only be obtained from the Human Interface manager. Connectors to the other resources are available through the Human Interface manager, or through the Console Manager in which the desired resource resides. Requests must specify the path-name of the resource as follows:

[<console_name>][/<screen_name>][/<window_or_ picture_ name>]

That is, the name of the console, optionally followed by a slash and the name of the screen, optionally followed by a slash and the name of a window, picture, or terminal. The console name may be omitted only if the message is sent directly to the desired console manager. If the screen name is omitted, the first screen configured on the given console is assumed. The window name must be specified if one of those resources is being connected.

Connection Requests

The "create" and "open" requests can be addressed to the "next" Human Interface context ("HI") or to a specific console connector or to the "next" context named "Console". If sent to "HI", a full path-name (the name parameter) must be given; otherwise, only the name of the desired resource is required (e.g., at a minimum, just the name of the window or picture).

If a picture manager process is created locally by an application, for private use, an "init" message—with the same contents as "create" or "open" must be sent directly to the picture process. The response will be "done" or "failed".

The following are the various Connection Requests and the types of information which may be associated with each:

CREATE is used to create a new picture resource, a new window resource, or a new virtual terminal resource.

When used to create a new picture resource, it may contain information about the resource type (i.e. a "picture"); the path-name of the picture; the size; the background color; the highlighting method; the maximum number of elements; the maximum element size; and the path-name of a library picture from which other elements may be copied.

When used to create a new window resource, it may contain information about the resource type (i.e. a "window"); the path-name of the window; the window's title; the window's position on the screen; the size of the window; the color, width, fill color between the outline and the pane, and the style of the main window outline; the color and width of the pane outline; a mapping of part of a picture into the window; a modification notation; a special character notation; various options; a "when" parameter requesting notification of various specified actions on/within the window; a title bar; a palette bar; vertical and horizontal scroll bars; a general use bar; and a corner box.

When used to create a new virtual terminal, it may contain information about the resource type (i.e. a "terminal"); the path-name of the terminal; the title of the terminal's window; various options; the terminal's position on the screen; the size of the terminal (i.e. number of lines and columns in the window); the maximum height and width of the virtual screen; the color the text inside the window; tab information; emulator process information; connector information to an existing window; window frame color; a list of menu items; and alternative format information.

OPEN is used to connect to a Human Interface service or to an existing Human Interface resource. When used to connect to a Human Interface service, it may contain information about the service type; and the name of the particular instance of the service. This resource must be sent to the Human Interface context.

When used to connect to an existing Human Interface resource, it may contain information about the path-name of the resource; the type of resource (e.g. picture, window, or terminal); and the name of the file (for pictures only) from which to load the picture. This request can be sent to a Human Interface manager or a console manager; alternatively the same message with message I.D. "init" specifying a file can be sent directly to a privately owned picture manager.

DELETE is used to remove an existing Human Interface resource from the system, and it may contain information specifying a connection to the resource; the type of resource; and whether, for a window, the corresponding picture is to be deleted at the same time.

CLOSE is used to break a connection to a Human Interface resource, and it may contain information specifying a connection to the resource; and the type of resource.

WHO? is used to request a list of signed-on users, and it may contain a user identification string.

QUERY is used to get the status of a service or resource, and it may contain information about the resource type; the name of the service or resource; a connector to a resource; and information concerning various options.

The following are the various Connection Responses and the types of information which may be associated with each:

CONNECT provides a connection to a Human Interface resource, and it contains information concerning the originator (i.e. the Human Interface or the console); the resource type; the original request message identifier; the name of the resource; and a connector to the resource.

USER contains the names of zero or more currently signed-on users and their locations, and it contains a connector to a console manager followed by the name of the user signed on at that console.

Console Requests

The main purpose of the console is to coordinate the activities of the windows, pictures, and dialog associated with it. Any of the CREATE, OPEN, DELETE, and CLOSE connection requests listed above, except those relating to the consoles, can be sent directly to a known console manager, rather than to the Human Interface manager (which always searches for the console by name). Subsequently, some characteristics of a window, such as its size, can be changed dynamically through the console manager. The current "user" of the console can be changed. And the console can be queried for its current status (or that of any of its resources).

The following are the various Console Requests and the types of information which may be associated with each:

USER is used to change the currently signed-on user, and it contains a user identification string.

CHANGE is used to change the size and other conditions of a window, and it may contain information about a connector to a window or a terminal; new height and width (in virtual pixels); increment to height and width; row and column position; various options; a connector to a new owner process; and whether the window should be the current active window on the screen.

CURSOR is used to move the screen cursor, and it contains position information as to row and column.

QUERY is used to get the current status of the console or one of its resources, and it contains information in the form of a connector to the resource; and various query options (e.g. list all screens, all pictures, or all windows).

BAND starts/stops the rubber-banding function and dragging function, and it contains information about the position of a point in the picture from which to start the operation; the end point of the figure which is to be dragged; the type of operation (e.g. line, rectangle, circle, or ellipse); the color; and the type of line (e.g. solid). In rubber-banding the drawn figure changes in size as the cursor is moved. In dragging the figure moves with the cursor.

The following are the various Console Responses and the types of information which may be associated with each:

STATUS describes the current state of a console, and it may contain information about a connector to the console; the originator; the name of the console; current cursor position; current metaphor size; scale of virtual pixels per centimeter, vertically and horizontally; number of colors supported; current user i.d. string; screen size and name; window connector and name; and picture connector, screen name, and window name.

Picture-Drawing

The picture is the fundamental building block in the Human Interface. It consists of a list of zero or more "picture elements", each of which is a device-independent abstraction of a displayable object (line, text, etc.). Each currently active picture is stored and maintained by a separate picture manager. "Drawing" a picture consists of sending picture manipulation messages to the picture manager.

A picture manager must first be initialized by a CREATE or OPEN request (or INIT, if the picture was created privately). CREATE sets the picture to empty, gives it a name, and defines the background. The OPEN request reads a predefined picture from a file and gives it a name. Either must be sent first before anything else is done. A subsequent OPEN reloads the picture from the file.

The basic request is to WRITE one or more elements. WRITE adds new elements to the end of the current list, thus

NUMBER contains element numbers and identifiers, and it may contain information specifying a list of numbers; and a list of element identifiers.

Picture Elements

Picture elements are defined by a collection of data structures, comprising one for a common "header", some optional structures, and one for each of the possible element types. The position of an element is always given as a set of absolute coordinates relative to [0,0] in the picture. This defines the upper left corner of the "box" which encloses each element. Points specified within an element (e.g. to define points on a line) are always given as coordinates relative to this position. In a "macro" the starting position of each individual element is considered to be relative to the absolute starting position of the macro element itself, i.e. they're nested.

FIG. 10 shows the general structure of a complete picture element. The "value" part depends upon the element type. The "appl" and "tag" fields are optional, depending upon indicators set in "attr".

The following is a description of the various fields in a picture element:

Length=length of the entire picture element in bytes

Type=one of the following: text, line, rectangle, ellipse, circle, symbol, array, discrete, macro, null, meta-element

Attr=one of the following: selectable, selected, rectilinear, inverted foreground/background, blink, tagged, application mnemonic, hidden, editable, movable, copyable, erasable, transformed, highlighted, mapped/not mapped, marked, copy

Pos=Row/col coordinates of upper left corner of the element's box

Box=Height/width of an imaginary box which completely and exactly encloses the element

Color=color of the element, consisting of 3 sub-fields: hue, saturation, and value

Bkgrnd=background color of the element

Fill=the color of the interior of a closed figure

Pattern=one of 10 "fill" patterns

Appl=a mnemonic referencing a particular application (e.g. forms manager, word-processor, report generator, etc.); allows multiple processes to share a single picture.

Tag=a variable-length, null-terminated string, supplied by the user; it can be used by applications to identify particular elements or classes of elements, or to store additional attributes

The attributes relating to the "type" field if designated "text" are as follows:

Options=wordwrap, bold, underline, italic, border, left-justify, right-justify, centered, top of box, bottom of box, middle of box, indent, tabs, adjust box size, character size, character/line spacing, and typeface

Select=indicates a currently selected substring by offset from beginning of string, and length

String=any number of bytes containing ASCII codes, followed by a single null byte; the text will be constrained to fit within the element's "box", automatically breaking to a new row when it reaches the right boundary of the area

Indent=two numbers specifying the indentation of the first and subsequent rows of text within the element's "box"

Tabs=list of [type, position], where "position" is the number of characters from the left edge of the element's box, and "type" is either Left, Right, or Decimal

Grow=maximum number of characters (horizontally) and lines (vertically) by which the element's box may be extended by typed input; limits growth right and downward, respectively

Size=height of the characters' extent and relative width

Space=spacing between lines of text and between characters

Face=name of a particular typeface

The attributes relating to the "type" field if designated "line" are as follows:

Style=various options such as solid, dashed, dotted, double, dashed-dotted, dash-dot-dot, patterned, etc.

Pattern=a pattern number

Thick=width of the line in pixels

Points=two or more pairs of coordinates (i.e. points) relative to the upper left corner of the box defined in the header

The attributes relating to the "type" field if designated "rectangle" are as follows:

Style=same as for "line" above, plus solid with a shadow

Pattern=same as for "line"

Thick=same as for "line"

Round=radius of a quarter-circle arc which will be drawn at each corner of the rectangle

The attributes relating to the "type" field if designated "ellipse" are as follows:

Style=solid, patterned, or double

Pattern=same as for "line"

Thick=same as for "line"

Arc=optional start- and end-angles of an elliptical arc

The attributes relating to the "type" field if designated "circle" are as follows:

Style=same as for "ellipse"

Pattern=same as for "line"

Thick—same as for "line"

Center=a point specifying the center of the circle, relative to the upper left corner of the element's box

Radius=length of the radius of the circle

Arc=optional start- and end-angles of a circular arc

A "symbol" is a rectangular space containing pixels which are visible (drawn) or invisible (not drawn). It is represented by a two-dimensional array, or "bit-map" of 1's and 0's with its origin in the upper left corner.

The attributes relating to the "type" field if designated "symbol" are as follows:

Bitmap=a two-dimensional array (in row and column order) containing single bits which are either "1" (draw the pixel in the foreground color) or "0" (draw the pixel in the background color); the origin of the array corresponds to the starting location of the element

Alt=A text string which can be displayed on non-bit-mapped devices, in place of the symbol

An array element is a rectangular space containing pixels which are drawn in specific colors, similar to a symbol element. It is represented as a two-dimensional array, or "bit-map", of color numbers, with its origin in the upper left corner. The element's "fill" and "pattern" are ignored.

The attributes relating to the "type" field if designated "array" are as follows:

Bitmap=a two-dimensional array (in row and column order) of color numbers; each number either defines a color in which a pixel is to be drawn, or is zero (in which the pixel is drawn in the background color); the origin of the array corresponds to the starting location of the element

Alt=an alternate text string which can be displayed on non-bit-mapped devices in place of the array

A discrete element is used to plot distinct points on the screen, optionally with lines joining them. Each point is specified by its coordinates relative to the element's "box" An explicit element (usually a single-character text element or a symbol element) may be given as the mark to be drawn at each point. If not, an asterisk is used. The resulting figure cannot be filled.

The attributes relating to the "type" field if designated "discrete" are as follows:

Mark=a picture element which defines the "mark" to be drawn at each point; if not applicable, a null-length element (i.e., a single integer containing the value zero) must be given for this field

Style Pat Thick =type, pattern, and thickness of the line (see "line" element above)

Join="Y" or "N" (or null, which is equivalent to "N"); if "Y", lines will be drawn to connect the marks

Points=two or more pairs of coordinates; each point is relative to the upper left corner of the "box" defined in the header

A "macro" element is a composite, made up of the preceding primitive element types ("text", etc.) and/or other macro elements. It can sometimes be thought of as "bracketing" other elements. The coordinates of the contained elements are relative to the absolute coordinates of the macro element. The "length" field of the macro element includes its own header and the "macro" field, plus the sum of the lengths of all of the contained elements. The "text" macro is useful for mixing different fonts and styles in single "unit" (word, etc.) of text.

The attributes relating to the "type" field if designated "macro" are as follows:

Macro=describes the contents of the macro element; may be one of following:

"N"—normal (contained elements are complete)

"Y"—list: same as "N", but only one sub-element at a time can be displayed; the others will be marked "hidden", and only the displayed element will be sent in response to requests ("copy", etc.); the "highlight" request will cycle through the sub-elements in order

"T"—text: same as "N", but the "macro" field is immediately followed by a text "options" field, and a text "select" field; the macro "list" field may be followed by further text parameters (as specified in the options field)

List=any number of picture elements (referred to as sub-elements), formatted as described above; terminated by a null word

A "meta-element" is a pseudo-element generated by the picture manager and which describes the picture itself, whenever the picture is "saved" to a file. Subsequently, meta-elements read from a file are used to set up parameters pertinent to the picture, such as its size and background color. Meta-elements never appear in "write" messages issued by the picture manager (e.g. in response to a "read" request, or as an update to a window manager).

The format of the meta-element includes a length field, a type field, a meta-type field, and a value. The 16-bit length field always specifies a length of 36. The type field is like that for normal picture elements. The meta-element field contains one of the following types:

Name=the value consists of a string which names the picture

Size=the maximum row and column, and the maximum element number and size

Backgnd—the picture's background color

Highlt=the picture's highlighting

The format of the value field depends upon the meta-type.

Windowing

A window maps a particular subset (often called a "view") of a given picture onto a particular screen. Each window on a screen is a single resource which handles the "pane" in which the picture is displayed and up to four "frame bars".

With reference to FIG. 11, a frame bar is used to show ancillary information such as a title. Frame bars can be interactive, displaying the names of "pull-down" menus which, when selected, display a list of options or actions pertinent to the window. A palette bar is like a permanently open menu, with all choices constantly visible.

Scroll bars indicate the relative position of the window's view in the picture and also allow scrolling by means of selectable "scroll buttons" A "resize" box can be selected to expand or shrink the size of the window on the screen while a "close" box can be selected to get rid of the window. Selecting a "blow-up" box expands the window to full screen size; selecting it again reduces it to its original dimensions.

A corner box is available for displaying additional user information, if desired.

The window shown in FIG. 11 comprises a single pane, four frame bars, and a corner box. The rectangular element within each scroll bar indicates the relative position of the window in the picture to which it is mapped (i.e. about a third of the way down and a little to the right).

Performing an action (such as a "select") in any portion of the window will optionally send a "click" message to the owner of the window. For example, selecting an element inside the pane will send "click" with "action"="select" and "area"="inside", as well as the coordinates of the cursor (relative to the top left corner of the picture) and a copy of the element at that position.

Selecting the name of a menu, which may appear in any frame bar, causes the menu to pop-up. It is the response to the menu that is sent in the "click" message, not the selection of the menu bar item. Pop-up menus (activated by menu keys on the keyboard) and function keys can also be associated with a particular window.

All windows are created by sending a "create" request to a Console Manager. As described above, "create" is the most complex of the windowing messages, containing numerous options which specify the size and location of the window, which frame bars to display, what to do when certain actions are performed in the window, and so on.

The process which sent the request is known as the "owner" of the window, although this can be changed dynamically. The most recently opened window usually becomes the current "active" window, although this may be overridden or changed.

What is claimed is:

1. A virtual input interface in a data processing system, said interface comprising:

means for accepting input from at least one physical device and for converting said physical device input into virtual input, said means comprising a virtual input manager process responsive to said at least one physical input device for generating a picture, said picture comprising one or more picture elements, each picture element comprising a plurality of device-independent data structures in a predetermined, standard data format, at least one of said data structures comprising a plurality of different data fields each containing information describing said picture element; and

means responsive to said virtual input for performing processing operations upon said virtual input, said means comprising a console manager process for performing processing operations on said one or more picture elements.

2. The virtual input interface as recited in claim 1, wherein said input accepting means accepts input in the form of keystrokes.

3. The virtual input interface as recited in claim 1, wherein said input accepting means accepts input in the form of cursor position.

4. The virtual input interface as recited in claim 1, wherein said input accepting means accepts input in the form of system-defined actions.

5. The virtual input interface as recited in claim 1, wherein said input accepting means accepts input in the form of user-defined functions.

6. The virtual input interface as recited in claim 1, wherein said input accepting means accepts input in the form of menu selections.

7. The virtual input interface as recited in claim 1, wherein said at least one physical device can be removed from said system without affecting the operation of the remainder of said system.

8. The virtual input interface as recited in claim 1, wherein at least one additional physical device can be added to said system without affecting the operation of the remainder of said system.

9. A virtual output interface in a data processing system, said interface comprising:

a source of virtual input, said virtual input comprising one or more picture elements, each picture element comprising a plurality of device-independent data structures in a predetermined, standard data format, at least one of said data structures comprising a plurality of different data fields each containing information describing said picture element;

means for performing processing operations on said virtual input and for generating virtual output;

means for accepting said virtual output; and

means for converting said virtual output into at least one physical output suitable for use by at least one physical output device.

10. The virtual output interface as recited in claim 9, wherein said virtual input comprises a plurality of related picture elements and wherein said virtual output accepting means comprises a picture manager process for controlling said plurality of related picture elements.

11. The virtual output interface as recited in claim 10 and further comprising a display device, wherein said virtual output accepting means further comprises a window manager process for controlling the display of said plurality of related picture elements on said display device.

12. The virtual output interface as recited in claim 9, wherein said virtual output converting means comprises a virtual output manager process responsive to said one or more processed picture elements for coupling said one or more processed picture elements to said at least one physical output device.

13. The virtual output interface as recited in claim 9, wherein said at least one physical device can be removed from said system without affecting the operation of the remainder of said system.

14. The virtual output interface as recited in claim 9, wherein at least one additional physical device can be added to said system without affecting the operation of the remainder of said system.

15. In a data processing system, an interface between processes and data in said system and physical input and output devices coupled to said system, said interface comprising:

means responsive to one of said physical input devices for generating a picture, said picture comprising one or more picture elements, each picture element comprising a plurality of device-independent data structures in a predetermined, standard data format, at least one of said data structures comprising a plurality of different data fields each containing information describing said picture element;

means for performing processing operations on said one or more picture elements; and

means responsive to said one or more processed picture elements for coupling said one or more processed picture elements to one of said physical output devices.

16. The data processing system as recited in claim 15, wherein said one or more picture elements define a graphical object and at least one attribute thereof.

17. The data processing system as recited in claim 16, wherein one of said data fields describes the length of the associated picture element.

18. The data processing system as recited in claim 16, wherein one of said data fields identifies the particular type of the associated picture element.

19. The data processing system as recited in claim 16, wherein one of said data fields describes the position of the associated picture element relative to row and column coordinates on a picture of which said picture element forms a part.

20. The data processing system as recited in claim 16, wherein one of said data fields describes the size of the associated picture element.

21. The data processing system as recited in claim 16, wherein one of said data fields describes the color of the associated picture element.

22. The data processing system as recited in claim 15, wherein said means responsive to one of said physical input devices comprises a virtual input manager process.

23. The data processing system as recited in claim 15, wherein said means responsive to said one or more processed picture elements comprises a virtual output manager process.

* * * * *