Motorola Mobility, Inc. v. Microsoft Corporation Doc. 123 Att. 17

EXHIBIT 38

Dockets.Justia.com

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

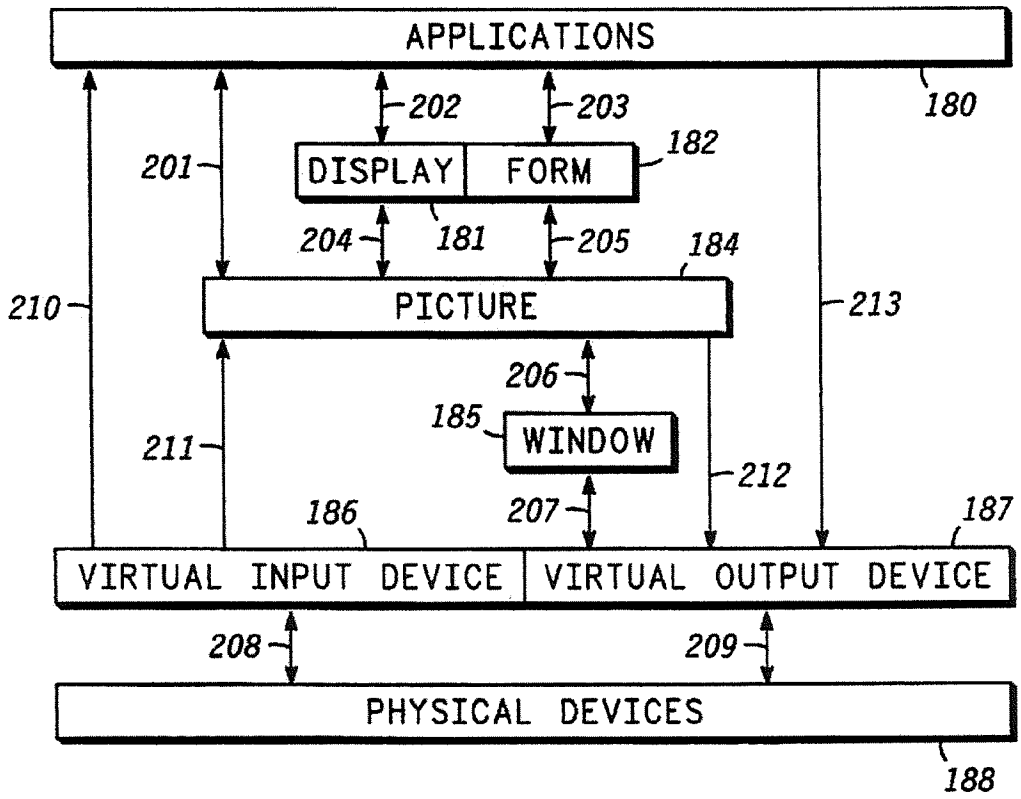
3,930,232	12/1975	Wallach et al.	395/500
4,241,341	12/1980	Thorson	340/747
4,454,593	6/1984	Fleming	364/900
4,485,439	11/1984	Rothstein	395/500
4,547,628	10/1985	Tamura et al.	340/747
4,555,775	11/1985	Pike	364/900
4,559,614	12/1985	Peek et al.	364/900
4,642,790	2/1987	Minshull et al.	364/900
4,754,395	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



CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

MOTM_24063_0019496

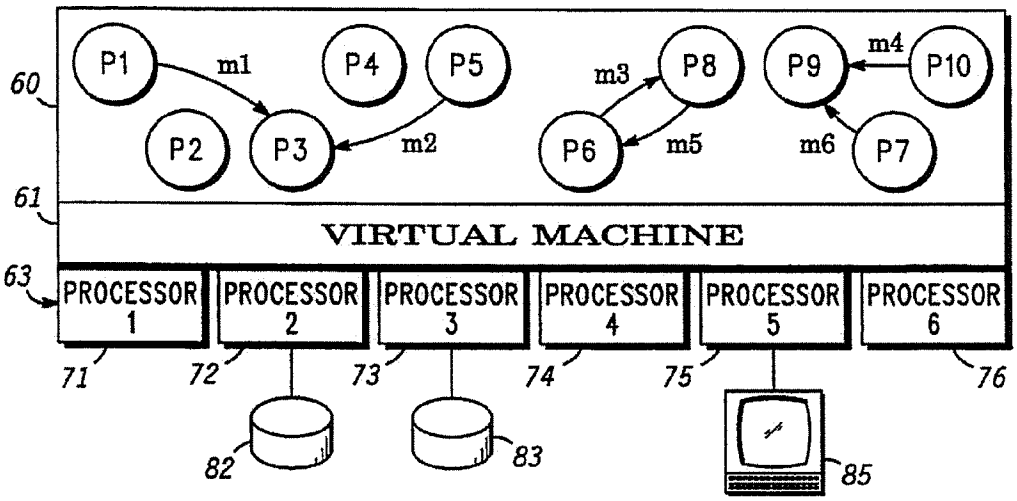
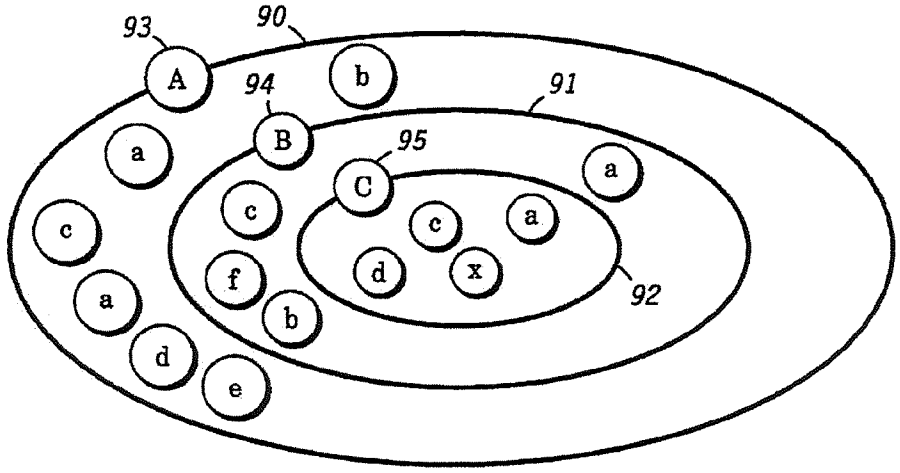


FIG. 3

FIG. 4



CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

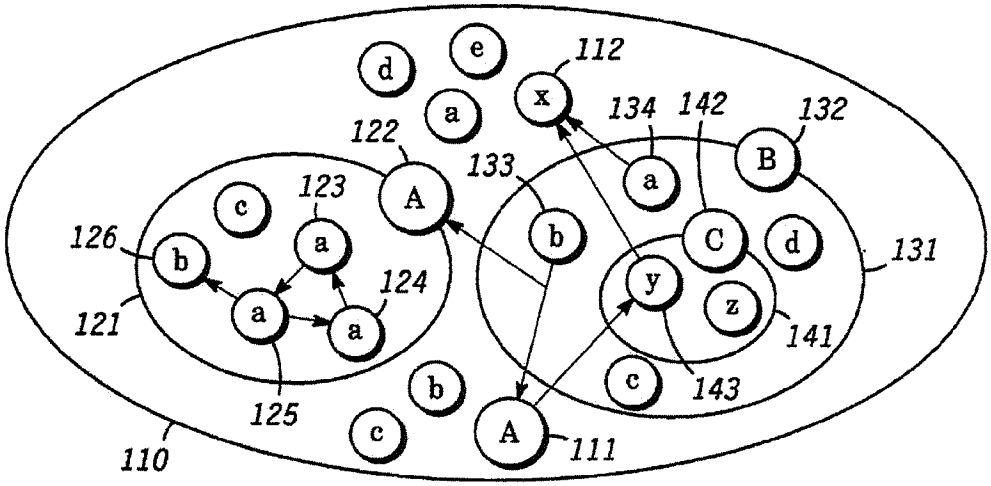
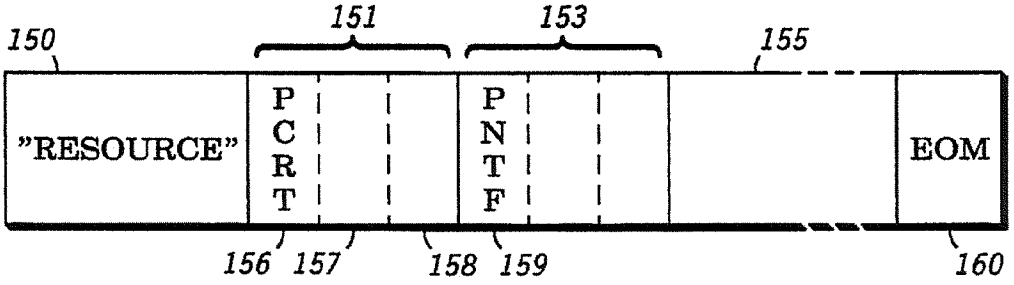


FIG. 5

FIG. 6



CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

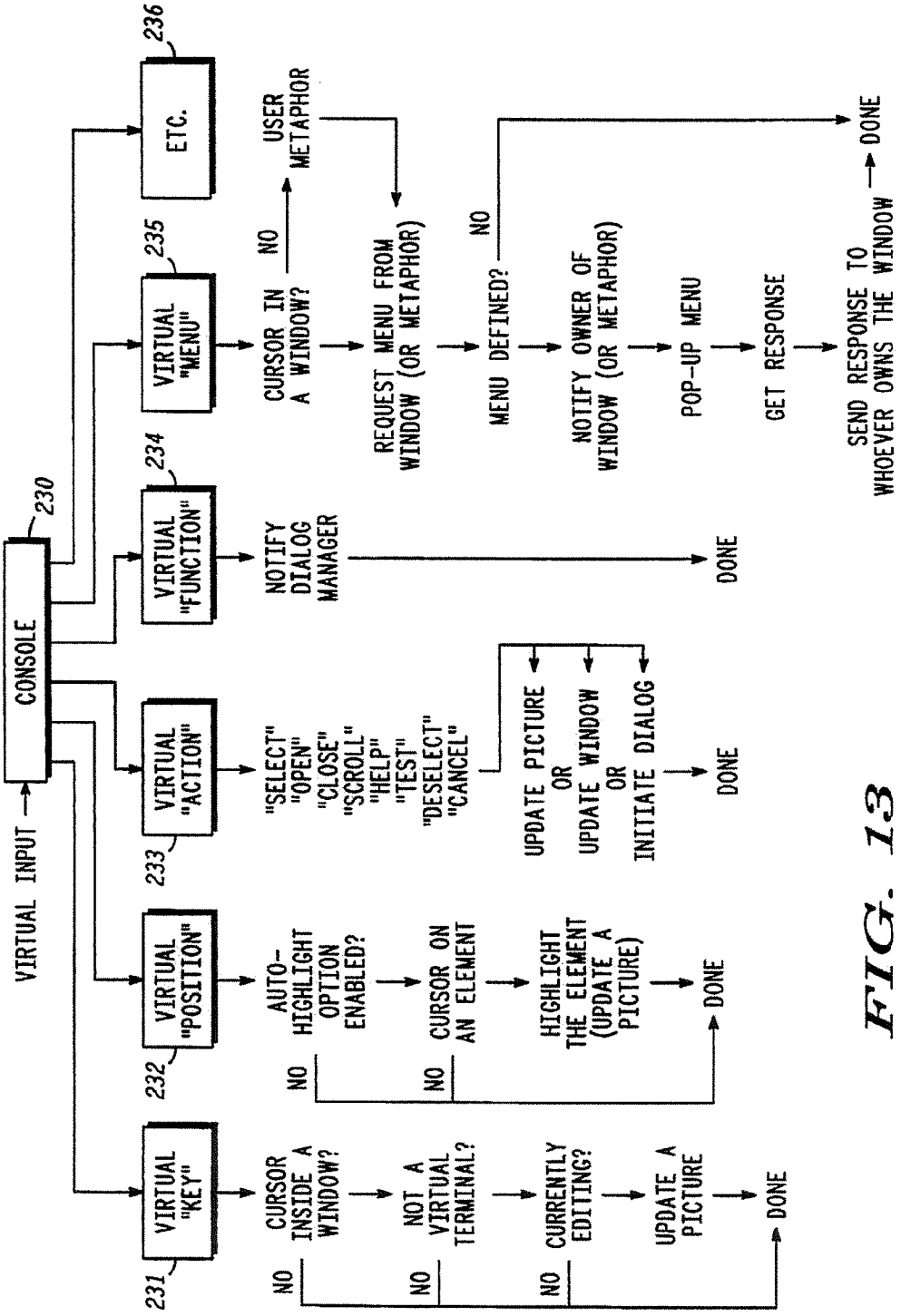


FIG. 13

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

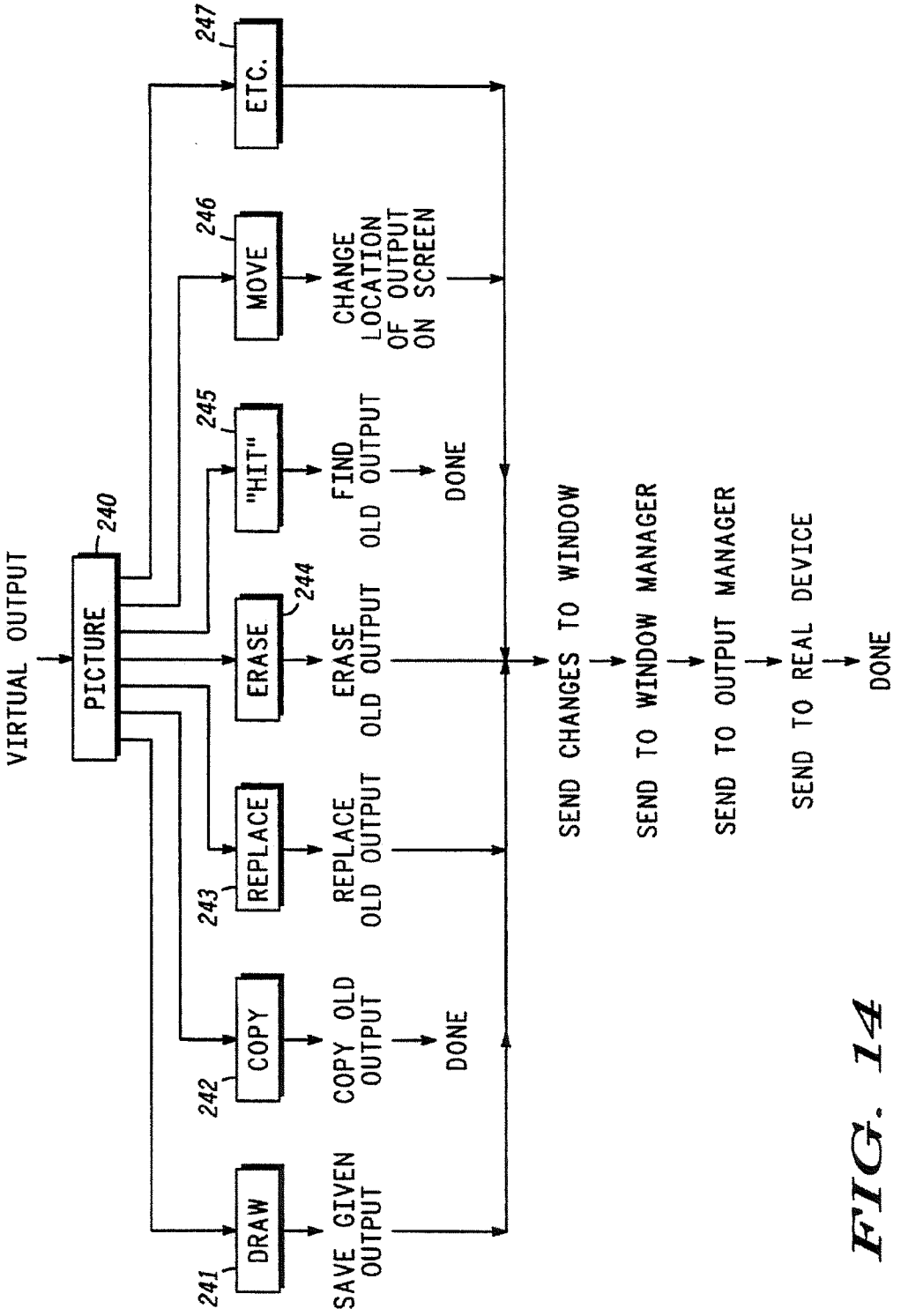


FIG. 14

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

pool which contains a linked list of pending "frames" Logical messages, which vary in length, are broken into fixed-size frames for transmission and are reassembled by the receiving NIM. Frames are sequence-numbered for this purpose. If a frame is not acknowledged within a short period of time, it is retransmitted a number of times before being treated as a failure.

As described above with reference to FIG. 2, the LAN may be connected to other LAN's operating under the same LAN protocol via so-called "bridgeways", or it may be connected to other types of LAN's via "gateways".

SOFTWARE MODEL

The computer operating system of the present invention operates upon processes, messages, and contexts, as such terms are defined herein. Thus this operating system offers the programmer a hardware abstraction, rather than a data or control abstraction.

A "process", as used within the present invention, is defined as a self-contained package of data and executable procedures which operate on that data, comparable to a "task" in other known systems. Within the present invention a process can be thought of as comparable to a subroutine in terms of size, complexity, and the way it is used. The difference between processes and subroutines is that processes can be created and destroyed dynamically and can execute concurrently with their creator and other "subroutines".

Within a process, as used in the present invention, the data is totally private and cannot be accessed from the outside, i.e., by other processes. Processes can therefore be used to implement "objects", "modules", or other higher-level data abstractions. Each process executes sequentially. Concurrency is achieved through multiple processes, possibly executing on multiple processors.

Every process in the distributed data processing system of the present invention has a unique identifier (PID) by which it can be referenced. The PID is assigned by the system when the process is created, and it is used by the system to physically locate the process.

Every process also has a non-unique, symbolic "name", which is a variable-length string of characters. In general, the name of a process is known system-wide. To restrict the scope of names, the present invention utilizes the concept of a "context".

A "context" is simply a collection of related processes whose names are not known outside of the context. Contexts partition the name space into smaller, more manageable subsystems. They also "hide" names, ensuring that processes contained in them do not unintentionally conflict with those in other contexts.

A process in one context cannot explicitly communicate with, and does not know about, processes inside other contexts. All interaction across context boundaries must be through a "context process", thus providing a degree of security. The context process often acts as a switchboard for incoming messages, rerouting them to the appropriate subprocesses in its context.

A context process behaves like any other process and additionally has the property that any processes which it creates are known only to itself and to each other. Creation of the process constitutes definition of a new context with the same name as the process.

Any process can create context processes. Each new context thus defined is completely contained inside the context in which it was created and therefore is shielded from outside reference. This "nesting" allows the name space to be structured hierarchically to any desired depth.

Conceptually, the highest level in the hierarchy is the system itself, which encompasses all contexts. Nesting is used in top-down design to break a system into components or "layers", where each layer is more detailed than the preceding one. This is analogous to breaking a task down into subroutines, and in fact many applications which are single tasks on known systems may translate to multiple processes in nested contexts.

A "message" is a buffer containing data which tells a process what to do and/or supplies it with information it needs to carry out its operation. Each message buffer can have a different length (up to 64 kilobytes). By convention, the first field in the message buffer defines the type of message (e.g., "read", "print", "status", "event", etc.).

Messages are queued from one process to another by name or PID. Queuing avoids potential synchronization problems and is used instead of semaphores, monitors, etc. The sender of a message is free to continue after the message is sent. When the receiver attempts to get a message, it will be suspended until one arrives if none are already waiting in its queue. Optionally, the sender can specify that it wants to wait for a reply and is suspended until that specific message arrives. Messages from any other source are not dequeued until after that happens.

Within the present invention, messages are the only way for two processes to exchange data. There is no concept of a "global variable". Shared memory areas are not allowed, other than through processes which essentially "manage" each area by means of messages. Messages are also the only form of dynamic memory that the system handles. A request to allocate memory therefore returns a block of memory which can be used locally by the process but can also be transmitted to another process.

The context nesting level determines the "scope of reference" when sending messages between processes by name. From a given process, a message may be sent to all processes at its own level (i.e., in the same context) and (optionally) to any arbitrary higher level. The contexts are searched from the current context upward until a match is found. All processes with the given name at that level are then sent a copy of the message. A process may also send a message to itself or to its parent (the context process) without knowing either name explicitly, permitting multiple instances of a process to exist in different contexts, with different names.

Sending messages by PID obviates the need for a name search and ignores context boundaries. This is the most efficient method of communicating.

Processes are referenced without regard to their physical location via a small set of message-passing primitives. As mentioned earlier, every process has both a unique system-generated identifier and a not necessarily unique name assigned by the programmer. The identifier provides quick direct access, while the name has a limited scope and provides symbolic, indirect access.

With reference to FIG. 3, an architectural model of the present invention is shown. The bottom, or hardware, layer **63** comprises a number of processors **71–76**, as described above. The processors **71–76** may exist physically within one or more nodes. The top, or software, layer **60** illustrates a number of processes P1–P10 which send messages m1–m6 to each other. The middle layer **61**, labelled "virtual machine", isolates the hardware from the software, and it allows programs to be written as if they were going to be

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

MOTM_24063_0019508

executed on a single processor. Conversely, programs can be distributed across multiple processors without having been explicitly designed for that purpose.

THE VIRTUAL MACHINE

As discussed earlier, a "process" is a self-contained package of data and executable procedures which operate on that data. The data is totally private and cannot be accessed by other processes. There is no concept of shared memory within the present invention. Execution of a process is strictly sequential. Multiple processes execute concurrently and must be scheduled by the operating system. The processes can be re-entrant, in which case only one copy of the code is loaded even if multiple instances are active.

Every process has a unique "process identifier number" (PID) by which it can be referenced. The PID is assigned by the system when the process is created and remains in effect until the process terminates. The PID assignment contains a randomizing factor which guarantees that the PID will not be re-used in the near future. The contents of the PID are irrelevant to the programmer but are used by the virtual machine to physically locate the process. A PID may be thought of as a "pointer" to a process.

Every process also has a "name" which is a variable-length string of characters assigned by the programmer. A name need not be unique, and this ambiguity may be used to add new services transparently and to aid in fault-tolerance.

FIG. 4 illustrates that the system-wide name space is partitioned into distinct subsets by means of "contexts" identified by reference numerals **90–92**. A context is simply a collection of related processes whose names are not known outside of the context. Context **90**, for example, contains processes A, a, a, b, c, d, and e. Context **91** contains processes B, a, b, c, and f. And context **92** contains processes C, a, c, d, and x.

One particular process in each context, called the "context process", is known both within the context and within the immediately enclosing one (referred to as its "parent context"). In the example illustrated in FIG. 4, processes A–C are context processes for contexts **90–92**, respectively. The parent context of context **91** is context **90**, and the parent context of context **92** is context **91**. Conceptually, the context process is located on the boundary of the context and acts as a gate into it.

Processes inside context **92** can reference any processes inside contexts **90** and **91** by name. However, processes in context **91** can only access processes in context **92** by going through the context process C. Processes in context **90** can only access processes in context **92** by going through context processes B and C.

The function of the context process is to filter incoming messages and either reject them or reroute them to other processes in its context. Contexts may be nested, allowing a hierarchy of abstractions to be constructed. A context must reside completely on one node. The entire system is treated as an all-encompassing context which is always present and which is the highest level in the hierarchy. In essence, contexts define localized protection domains and greatly reduce the chances of unintentional naming conflicts.

If appropriate, a process inside one context can be "connected" to one inside another context by exchanging PID's, once contact has been established through one or the other of the context processes. Most process servers within the present invention function that way. Initial access is by name. Once the desired function (such as a window or file) is "opened", the user process and the service communicate directly via PID's.

A "message" is a variable-length buffer (limited only by the processor's physical memory size) which carries information between processes. A header, inaccessible to the programmer, contains the destination name and the sender's PID. By convention, the first field in a message is a null-terminated string which defines the type of message (e.g., "read", "status", etc.) Messages are queued to the receiving process when they are sent. Queuing ensures serial access and is used in preference to semaphores, monitors, etc.

Messages provide the mechanism by which hardware transparency is achieved. A process located anywhere in the system may send a message to any other process anywhere else in the system (even on another processor) if it knows the process name. This means that processes can be dynamically distributed across the system at any time to gain optimal throughput without changing the processes which reference them. Resolution of destinations is done by searching the process name space.

Transparency applies with some restrictions across bridgeways (i.e., the interfaces between LAN's operating under identical network protocols) and, in general, not at all across gateways (i.e., the interfaces between LAN's operating under different network protocols) due to performance degradation. However, they could so operate, depending upon the required level of performance.

INTER-PROCESS COMMUNICATION

All inter-process communication is via messages. Consequently, most of the virtual machine primitives are concerned with processing messages. The virtual machine kernel primitives are the following:

- ALLOC—requests allocation of a (message) buffer of a given size.
- FREE—requests deallocation of a given message buffer.
- PUT—send a message to a given destination (by name or PID).
- GET—wait for and dequeue the next incoming message, optionally from a specific process (by PID).
- FORWARD—pass a received message through to another process.
- CALL—send a message, then wait for and dequeue the reply.
- REPLY—send a message to the originator of a given message.
- ANY_MSG—returns "true" if the receive queue is not empty, else returns "false"; optionally, checks if any messages from a specific PID are queued.

To further describe the function of the kernel primitives, ALLOC handles all memory allocations. It returns a pointer to a buffer which can be used for local storage within the process or which can be sent to another process (via PUT, etc.). ALLOC never "fails", but rather waits until enough memory is freed to satisfy the request.

The PUT primitive queues a message to another process. The sending process resumes execution as soon as the message is queued.

FORWARD is used to quickly reroute a message but maintain information about the original sender (whereas PUT always makes the sending process the originator of the message).

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

MOTM_24063_0019509

REPLY sends a message to the originator of a previously received message, rather than by name or PID.

CALL essentially implements remote subroutine invocations, causing the caller to suspend until the receiver executes a REPLY. Subsequently, the replied message is dequeued out of sequence, immediately upon arrival, and the caller resumes execution.

The emphasis is on concurrency, so that as many processes as possible are executed in parallel. Hence neither PUT nor FORWARD waits for the message to be delivered. Conversely, GET suspends a process until a message arrives and dequeues it in one operation. The ANY MSG primitive is provided so that a process may determine whether there is anything of interest in the queue before committing itself to a GET.

When a message is sent by name, the destination process must be found in the name space. The search path is determined by the nesting of the contexts in which the sending process resides. From a given process, a message can be sent to all processes in its own context or (optionally) to those in any higher context. Refer to FIG. **5**. The contexts are searched from the current one upward until a match is found or until the system context is reached. All processes with the same name in that context are then queued a copy of the message.

For example, with reference to FIG. **5**, assume that in context **141** process y sends a message to ALL processes by the name x. Process y first searches within its own context **141** but finds no process x. The process y searches within the next higher context **131** (its parent context) but again finds no process x. Then process y searches within the next higher context **110** and finds a process x, identified by reference numeral **112**. Since it is the only process x in context **110**, it is the only recipient of the message from process y.

If process a in context **131** sends a message to ALL processes by the name x, it first searches within its own context **131** and, finding no processes x there, it then searches within context **110** and finds process x.

Assume that process b in context **131** sends a message to ALL processes by the name A. It would find process A (**111**) in context **110**, as well as process A (**122**) which is the context process for context **121**.

A process may also send a message to itself or to its context process without knowing either name explicitly.

The concept of a "logical ring" (analogous to a LAN) allows a message to be sent to the NEXT process in the system with a given name. The message goes to exactly one process in the sender's context, if such a process exists. Otherwise the parent context is searched.

The virtual machine guarantees that each NEXT transmission will reach a different process and that eventually a transmission will be sent to the logically "first" process (the one that sent the original message) in the ring, completing the loop. In other words, all processes with the same name at the same level can communicate with each other without knowing how many there are or where they are located. The logical ring is essential for distributing services such as a data base. The ordering of processes in the ring is not predictable.

For example, if process a (**125**) in context **121** sends a message to process a using the NEXT primitive, the search finds a first process a (**124**) in the same context **121**. Process a (**124**) is marked as having received the message, and then process a (**124**) sends the message on to the NEXT process a (**123**) in context **121**. Process a (**123**) is marked as having received the message, and then it sends the message on to the NEXT process a, which is the original sender process a (**125**), which knows not to send it further on, since it's been marked as having already received the message.

Sending messages directly by PID obviates the need for a name search and ignores context boundaries. This is known as the DIRECT mode of transmission and is the most efficient. For example, process A (**111**) sends a message in the DIRECT mode to process y in context **141**.

If a process sends a message in the LOCAL transmission mode, it sends it only to a process having the given name in the sender's own context.

In summary, including the DIRECT transmission mode, there are five transmission modes which can be used with the PUT, FORWARD, and CALL primitives:

ALL—to all processes with the given name in the first context which contains that name, starting with the sender's context and searching upwards through all parent contexts.

LOCAL—to all processes with the given name in the sender's context only.

NEXT—to the next process with the given name in the same context as the sender, if any; otherwise it searches upwards through all parent contexts until the name is found.

LEVEL—sends to "self" (the sending process) or to "context" (the context process corresponding to the sender's context); "self" cannot be used with CALL primitive.

DIRECT—sent by PID.

Messages are usually transmitted by queueing a pointer to the buffer containing the message. A message is only copied when there are multiple destinations or when the destination is on another node.

OPERATING SYSTEM

The operating system of the present invention consists of a kernel, which implements the primitives described above, plus a set of processes which provide process creation and termination, time management (set time, set alarm, etc.) and which perform node start-up and configuration. Drivers for devices are also implemented as processes (EESP's), as described above. This allows both system services and device drivers to be added or replaced easily. The operating system also supports swapping and paging, although both are invisible to applications software.

Unlike known distributed computer systems, that of the present invention does not use a distinct "name server" process to resolve names. Name searching is confined to the kernel, which has the advantage of being much faster.

A minimal bootstrap program resides permanently (in ROM) on every node, e.g. ROM **28** in node N of FIG. **2**. The bootstrap program executes automatically when a node is powered up and begins by performing basic on-board diagnostics. It then attempts to find and start an initial system code module. The module is sought on the first disk drive on the node, if any. If there isn't a disk, and the node is on the LAN, a message will be sent out requesting the module. Failing that, the required software must be resident in ROM. The initialization program of the kernel sets up all of the kernel's internal tables and then calls a predefined entry point of the process.

In general, there exists a template file describing the initial software and hardware for each node in the system. The template defines a set of initial processes (usually one per service) which are scheduled immediately after the node

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

MOTM_24063_0019510

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

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

MOTM_24063_0019512

"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

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

MOTM_24063_0019513

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

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

MOTM_24063_0019514

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

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

MOTM_24063_0019515

about the Window Manager: the Console Manager in the same context.

The Window Manager's main job is to copy picture elements from a given rectangular area of a picture to a rectangular area (called a "window") on a particular screen. To do so it interacts with exactly one Picture Manager and one Output Manager. A Window Manager need only be created when a window is "opened" on the screen and can be told to quit when the window is "closed" (without affecting the associated picture). When opened, the Window Manager must draw the outline, frame, and background of the window. When closed, the window and its frame must be erased (i.e. redrawn in the screen's background color and pattern). "Moving" a window (changing its location on the screen) is essentially the same as closing and re-opening it.

A Window Manager can only be created and destroyed by a Console Manager, which is responsible for arranging windows on the screen, resolving overlaps, etc. When a Window Manager is created, it waits for an "initialize" message, initializes itself, returns an "I'm here" message to the process which sent it the "initialize" message, then waits for further messages. It does not send any messages to the Output Manager until it has received all of the following: its dimensions (exclusive of frame), the outline line-type, size and color, background color, location on the screen, a clipping polygon, scaling factors, and framing parameters. A Window Manager also has an "owner", which is a particular process which will handle commands (through the Console Manager, which always has prime control) within the window.

Any of the above parameters can be changed at any time. In general, changing any parameter (other than the owner) causes the window to be redrawn on the screen.

A "frame", which may consist of four components (called "bars"), one along each edge of the window, may be placed around the given window. The bars are designated top, bottom, left, and right. They can be any combination of simple line segment, title bar, scroll bar, menu bar, and palette bar. These are supplied to the message as four separate lists (in four separate messages) of standard picture elements, whcih can be changed at any time by sending a new message referencing the bar. The origin of each bar is [0,0] relative to the upper left corner of the window.

The Console Manager may query a Window Manager for any of its parameters, to which it responds with messages identical to the ones it originally received. It can also be asked whether a given absolute cursor position is inside its window (i.e. inside the current clipping polygon) or its frame, and for the cursor coordinates relative to the origin of the window or any edge of the frame.

A Window Manager is tightly coupled to its creator (a Console Manager), Picture Manager, and Output Manager; i.e. they communicate with each other using process identifiers (PID's). Consequently, a Window Manager must inform its Picture Manager when it exits, and it expects the Picture Manager to do the same.

Once the Window Manager knows the picture it is accessing and the dimensions of its window (or any time either of these changes), it requests the Picture Manager to send it all picture elements which completely or partially lie within the window. It also asks it to notify it of chanages which will affect the displayed portion of the picture. The Picture Manager will send "draw" messages to the Window Manager (at any time) to satisfy these requests.

The Window Manager performs gross clipping on all picture elements it receives, i.e. it just determines whether each element could appear inside the current clipping polygon (which may be smaller than the window at any given moment, if other windows overlap this one).

A Window Manager can be told to "freeze" (stop updating) its display and to "unfreeze" it. It can also be asked to redraw any given rectangular sub-area of the picture it is displaying.

Window Managers deal strictly in virtual pixels and have no knowledge about the physical characteristics of the screen to which they are writing. Consequently, a window's size and location are specified in virtual pixels, implying a conversion from real pixels if these are different.

Print Manager—There is one per "output subsystem", i.e. per pool of output devices. The Print Manager coordinates output to hard-copy devices (i.e. to their Output Managers). It provides a comprehensive queueing service for files that need to be printed. It can also perform some minimal formatting of text (justification, automatic page numbering, headers, footers, etc.)

The Print Manager knows about the following processes: Output Managers in the same context, and a Picture Manager in the same context. The following processes know about the Print Manager: any one that wants to.

One Print Manager is created automatically, at start-up time, in each Print context. It is expected to accept general requests for hard-copy output and pass them on, one message (usually corresponding to one "line" of output) at a time, to the appropriate Output Manager. It can also accept requests which refer to files (i.e. to File Manager processes). Each such message, known as a "spool" request, also contains a priority, the number of copies desired, specific output device requirements (if any) and special form requirements (if any).

Based on these parameters, as well as the size of the file, the amount of time the request has been waiting, and the availability of output devices, the Print Manager maintains an ordered queue of outstanding requests. It dequeues them one at a time, select an Output Manager, and builds a picture (using a Picture Manager). It then requests (from the Picture Manager) and "prints" (plots, etc.) one "page" at a time until the entire file has been printed.

The Print Manager recognizes specially marked ("tagged") picture elements which define headers, footers, foot-notes, and page formatting parameters (such as "page break", "set page number", etc.).

HUMAN INTERFACE—RELATIONSHIPS BETWEEN COMPONENTS

The eight Human Interface components together provide all of the services required to support a minimal human interface. The relationships between them are illustrated in FIG. **9**, which shows at least one instance of each component. The components represented by circles **301, 302, 307, 312, 315**, and **317–320** are generally always present and active, while the other components are created as needed and exit when they have finished their specific funtions. FIG. **9** is divided into two main contexts: "Console" **350** and "Print" **351**.

Cursor **314** and Input **311** are examples of processes whose primary function is to store data. "Cursor"'s purpose is to keep track of the current cursor position on the screen and all parameters (such as the symbols defining different cursors) pertinent to the cursor. One cursor process is created by the Console Manager for each Output Manager when it is initialized. The Output Manager is responsible for updating the cursor data, although "Cursor" may be queried by anyone. "Input" keeps track of the current input state, such

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

MOTM_24063_0019517

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 param-

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

MOTM_24063_0019518

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.

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

MOTM_24063_0019519

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

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

MOTM_24063_0019520

reflecting the order. Whenever parts of the picture are copied or displayed, this order is preserved. Once drawn, one or more elements can be moved, erased, copied, or replaced. All or part of the picture can be saved to a given file. In addition, there are requests to quickly change a particular attribute of one or more elements (e.g. select them). Finally, the DELETE request (to the console manager; QUIT, if direct to the picture resource) terminates the picture manager, without saving the picture.

Any single element can be "marked" for later reference. If the element is text, then a particular offset in the string can be marked, and a visible mark symbol displayed at that location.

A picture can be shared among several processes ("applications") by setting the "appl" field in the picture elements. Each application process can treat the picture as if it contains only its own elements. All requests made by each process will only affect elements which contain a matching "appl" field. Participating processes must be identified to the picture manager via an "appl" request.

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

WRITE is used to add new elements to a picture, and it may contain information providing a list of picture elements; the data type; and an indication to add the new elements after the first element found in a given range (instead of the foreground, at the end of the list).

READ is used to copy elements from a picture, and it may contain information regarding the connection to which to send the elements; an indication to copy background elements; and a range of elements to be copied.

MOVE is used to move elements to another location, and it may contain information indicating a point in the picture to which the elements are to be moved; row and column offsets; to picture foreground; to picture background; fixed size increments; and a range of elements to be moved.

REPLACE is used to replace existing elements with new ones, and it may contain information providing a list of picture elements; and a range of elements to be replaced.

ERASE is used to remove elements from a picture, and it may contain information on the range of elements to be erased.

QUIT is used to erase all elements and terminate, and it has no particular parameters (valid only if the picture is private).

MARK is used to set a "marked" attribute (if text, to display a mark symbol), and it may contain information specifying the element to be marked; and the offset of the character after which to display the mark symbol.

SELECT is used to select an element and mark it, and it may contain information specifying the element(s) to be selected; the offset of the character after which to display the mark symbol; the number of characters to select; and a deselect option.

SAVE is used to copy all or part of a picture to a file, and it may contain information specifying the name of the file; and a subset of a picture.

QUERY is used to get the current status, and it has no particular parameters.

BKGD is used to change a picture's background color, and it may contain information specifying the color.

APPL is used to register a picture as an "application", and it may contain information specifying a name of the application; a connection to the application process; and a point of origin inside the picture.

NUMBER is used to get ordinal numbers and identifiers of specific elements, and it may contain information specifying the element(s).

HIT is used to find an element at or closest to a given position, and it may contain a position location in a picture; and how far away from the position the element can be.

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

HIGHLIGHT, INVERT, BLINK, HIDE are used to change a specific element attribute, and they may contain information indicating whether the attribute is set or cleared; and a range of elements to be changed.

CHANGE is used to change one or more element fields, and it may contain information specifying the color of the element; the background color; the fill color; and fill pattern; and a range of elements to be changed.

EDIT is used to modify a text element's string, and it may contain information indicating to edit at the current mark and then move the mark; specifying the currently selected substring is to be edited; an offset into the text at which to insert and/or from which to start shifting; to shift the text by the given number of characters to/from the given position; tab spacing; a replacement substring; to blank to the end of the element; and a range of elements to be edited.

In general, when a range of elements is specified, a list of one or more parameters is provided (if omitted, then all elements in the picture are referenced by default) according to the following table:

Keyword	Meaning	Format
@pos	by position (start of range)	row, column
@end	last position of a range	row, column
@num	by relative element number	list of numbers
@tag	search for a tag	pattern
@txt	search for a text element	pattern
@sel	"selected" element(s)	keyword only
@mrk	"marked" element	keyword only
@id	by unique element identifier	list of identifiers
@att	by attributes	attribute structure
@cnt	the number of elements	count

Any range parameters which are given restrict the elements which will be affected by the current request. In general, only the intersection of all of the elements satisfying the given conditions are included in the range. For example, specifying pos, end, tag, txt, and sel together means "use all selected text elements between the given coordinates, containing a particular tag and an particular text string.

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

STATUS describes the current status of the picture, and it may contain information specifying a connector to the picture; an original message identifier, if applicable; the name of the picture; the name of the file last read or written; height and width; lowest and highest row/column in the picture; the number of elements; and the number of currently active viewports.

WRITE contains elements copied from a picture, and it may contain information specifying a connector to the picture; a list of picture elements; and the data type.

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

MOTM_24063_0019521

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

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

MOTM_24063_0019527

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.

* * * * *

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

MOTM_24063_0019618

EXHIBIT 39

US007493130B2

(12) **United States Patent**
Loveland

(10) **Patent No.: US 7,493,130 B2**
(45) **Date of Patent: *Feb. 17, 2009**

(54) **SYNCHRONIZING OVER A NUMBER OF SYNCHRONIZATION MECHANISMS USING FLEXIBLE RULES**

(75) Inventor: **Shawn Domenic Loveland**, Sammamish, WA (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 368 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **11/340,346**

(22) Filed: **Jan. 26, 2006**

(65) **Prior Publication Data**

US 2006/0235898 A1 Oct. 19, 2006

Related U.S. Application Data

(63) Continuation of application No. 10/082,918, filed on Feb. 26, 2002, now Pat. No. 7,024,214.

(51) **Int. Cl.**
H04B 7/00 (2006.01)

(52) **U.S. Cl.** **455/502**; 455/501; 455/503; 455/500; 370/350; 370/503

(58) **Field of Classification Search** 455/502, 455/501, 503, 500; 370/350, 503, 506
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,308,223	B1 *	10/2001	Opgenoorth	709/248
7,024,214	B2 *	4/2006	Loveland	455/502
2003/0125057	A1 *	7/2003	Pesola	455/502

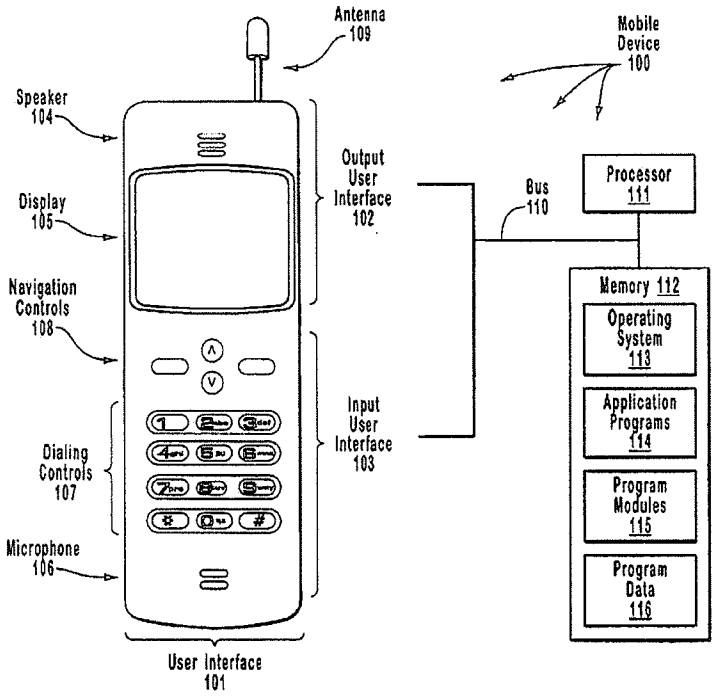
* cited by examiner

Primary Examiner—Vincent P. Harper
Assistant Examiner Khai M Nguyen

(57) **ABSTRACT**

Two computer systems in a network each have a local store that contains a copy of a data item that is to be synchronized. One of the computer systems may be, for example, a mobile device while the other may be a synchronization server. In order to determine whether to synchronize a data item, and what synchronization mechanism to use, one of the computer systems references a flexible set of rules that may be influenced by instructions from a network administrator or a mobile device user. The flexible set of rules takes into consideration the value of the data, the cost associated with synchronization, the security of the synchronization mechanisms, the security of the mobile device, as well as the location of the mobile user in dictating whether and how to synchronize.

20 Claims, 3 Drawing Sheets



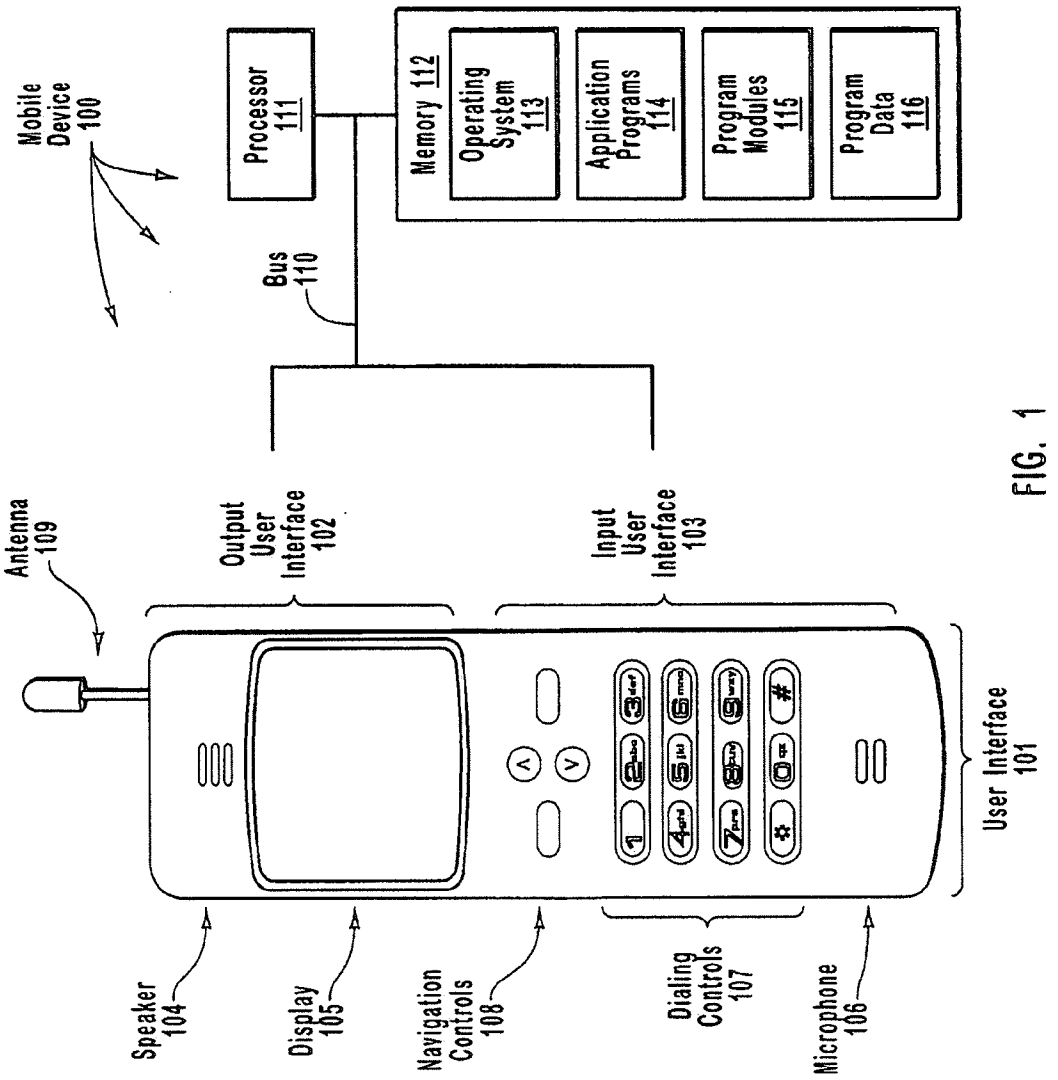

Mobile Device 100
Processor 111
Memory 112
Operating System 113
Application Programs 114
Program Modules 115
Program Data 116
Bus 110
Antenna 109
Output User Interface 102
Input User Interface 103
User Interface 101
Speaker 104
Display 105
Navigation Controls 108
Dialing Controls 107
Microphone 106

FIG. 1

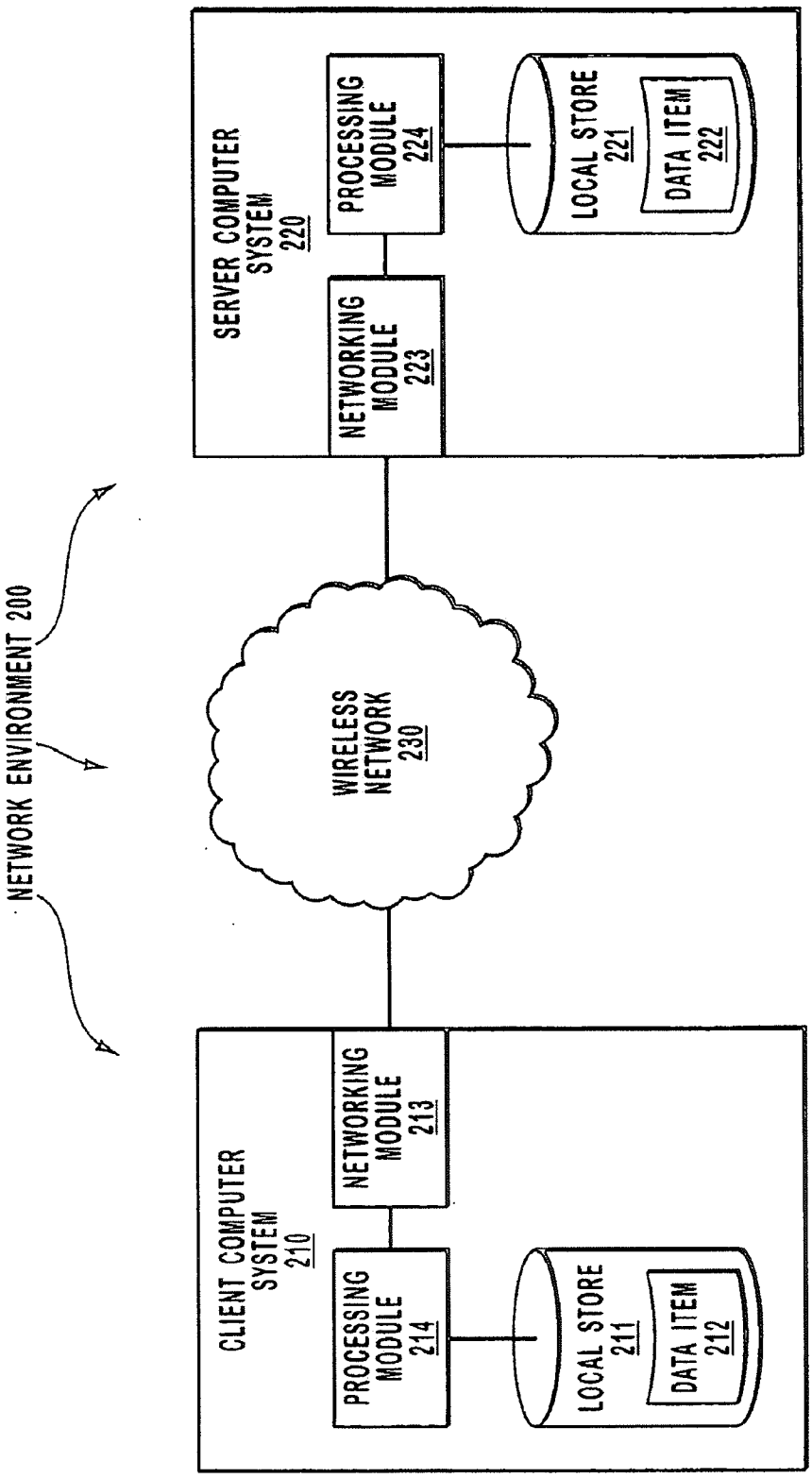

NETWORK ENVIRONMENT 200
SERVER COMPUTER SYSTEM 220
PROCESSING MODULE 224
LOCAL STORE 221
DATA ITEM 222
NETWORKING MODULE 223
WIRELESS NETWORK 230
CLIENT COMPUTER SYSTEM 210
NETWORKING MODULE 213
PROCESSING MODULE 214
LOCAL STORE 211
DATA ITEM 212

FIG. 2

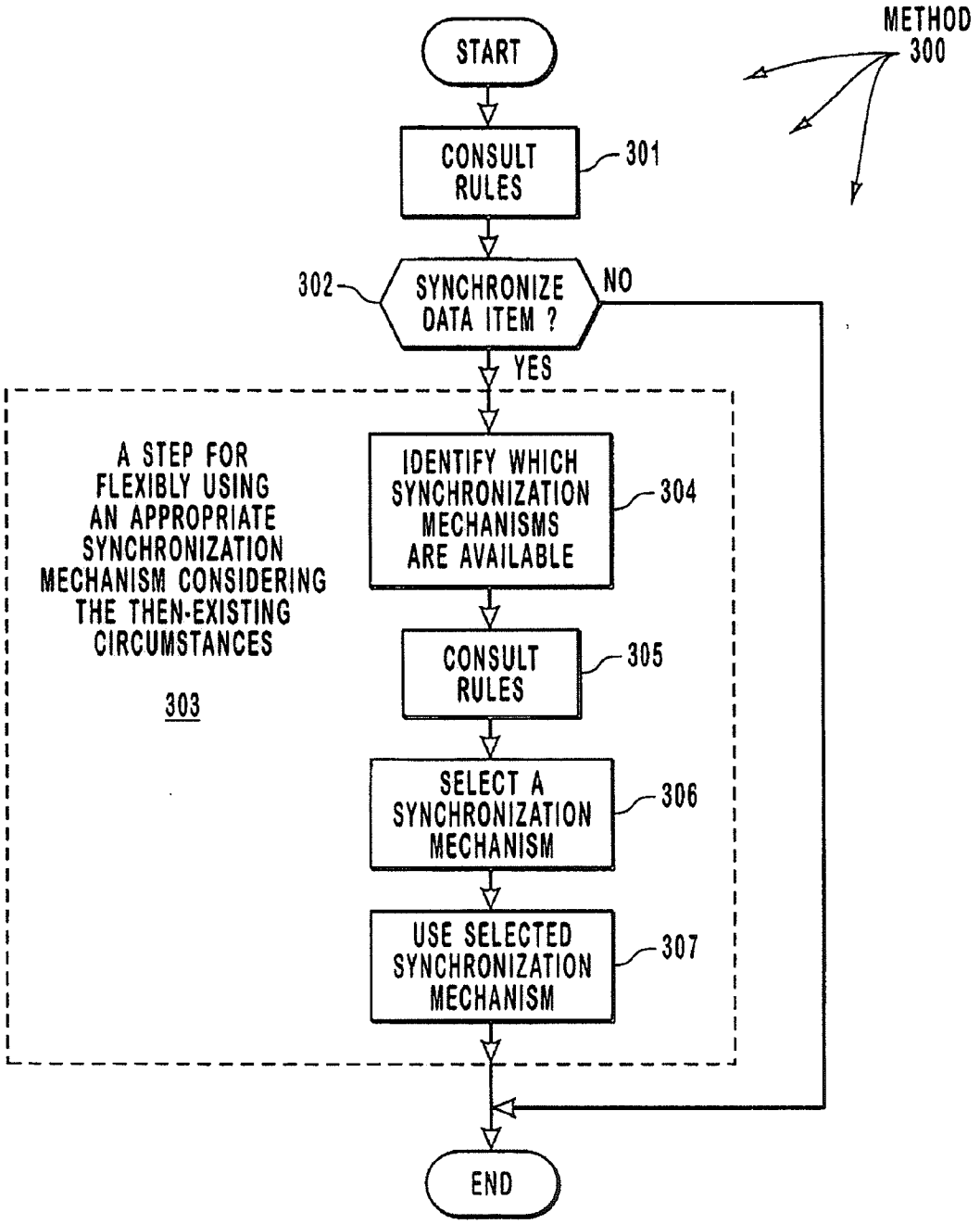

METHOD
300
START
CONSULT RULES
301
302
SYNCHRONIZE DATA ITEM ?
NO
YES
A STEP FOR FLEXIBLY USING AN APPROPRIATE SYNCHRONIZATION MECHANISM CONSIDERING THE THEN-EXISTING CIRCUMSTANCES
303
IDENTIFY WHICH SYNCHRONIZATION MECHANISMS ARE AVAILABLE
304
CONSULT RULES
305
SELECT A SYNCHRONIZATION MECHANISM
306
USE SELECTED SYNCHRONIZATION MECHANISM
307
END

FIG. 3

SYNCHRONIZING OVER A NUMBER OF SYNCHRONIZATION MECHANISMS USING FLEXIBLE RULES

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is a continuation of U.S. patent application Ser. No. 10/082,918, filed Feb. 26, 2002, and entitled "SYNCHRONIZING OVER A NUMBER OF SYNCHRONIZATION MECHANSIMS USING FLEXIBLE RULES", and which is incorporated herein by reference.

BACKGROUND OF THE INVENTION

1. The Field of the Invention

The present invention relates to computer network operations. In particular, the present invention relates to methods, systems, and computer program products for synchronizing by flexibly using multiple synchronization mechanisms while considering the then-existing economic and/or security considerations involved with synchronization over a particular mechanism to a particular device.

2. Background and Relevant Art

Computer networks allow more individuals more ready access to more information than ever before. The Internet is a conglomerate of interconnected computer networks that spreads far and wide throughout the world. An individual need only have an Internet-enabled computer (or device) and an Internet connection to be able to access information from across the globe.

Mobile devices such as mobile telephones; Personal Digital Assistants (PDAs) and laptop computers also may have the ability to access various objects (e.g., documents) for particular remote network locations. However, accessing documents over a network may be quite time consuming and costly and thus remote access may be unacceptable in many applications. Accordingly, with the memory capabilities of mobile devices increasing, many mobile devices have caching mechanisms that allow them to locally store synchronized copies of more relevant objects that are also stored in another network location.

Synchronization ensures that the copy of a document on the mobile device is an identical replica of a remote copy of the document. After synchronization, however, changes may be made to both of the remote copy or the local copy. Accordingly, the local and remote copies may become quite different after some time. However, the copies are once again made identical during the next synchronization.

Conventional synchronization mechanisms give the user very little control over synchronization. Basically, the user just selects items to synchronize and selects a single synchronization mechanism to use. This rigid approach ignores some factors that are relevant to whether, when, and how to synchronize.

For example, some data is more valuable than others. Yet, the conventional technique does not consider the value of the data once the items for synchronization are selected. For example, a user may select to synchronize an in-box. However, the in-box contains high-value e-mails (such as an e-mail offering a highly coveted job) as well as low value e-mails (such as unwanted spam). Yet, all of the e-mails are synchronized in the exact same manner.

Also current synchronization mechanisms do not consider the costs associated with synchronization. For example, synchronization mechanisms may involve networks with a wide variety of costs and latencies. For example, synchronization over an analog dialup that uses a Global System for Mobile communication (GSM) network has a relatively low bandwidth of approximately 9.6 to 14.4 kilobits per second. A connection over a General Packet Radio Server (GPRS) network allows somewhat faster throughput speeds of up to 115 kilobits per second. However, current GPRS networks are relatively expensive. A connection over an 802.11b (also called "WiFi") wireless network can attain speeds of up to 11 megabits per second, orders of magnitude greater than GSM or GPRS. In addition, use of 802.11b networks is typically free to those who have access to the network. However, 802.11b networks are typically less available as they currently span much less of the globe than do GSM or GPRS networks.

Current synchronization techniques also typically do not consider security concerns associated with synchronization. For example, the channel used for synchronization may have various levels of inherent security that guard against eavesdropping. For example, physical network connections are very secure, 802.11b networks are somewhat secure, with GSM dialup networks being somewhat less secure, and with GPRS networks being less secure. Also, the devices to which data may be synchronized may have varying levels of security. For example, some devices give access to any locally stored document to anyone who happens to possess the device, even if that person just stole the device from the legitimate user. Other devices are more secure by requiring a password when first logging in. Others are even more secure by having the screen saver turn on after a short period of lack of use, and require a password to restore the device once the screen saver is on. Others are yet even more secure by having an encrypted file system.

Accordingly, what is desired are mechanisms that use a synchronization mechanism that is appropriate given the economic and security concerns that exist at the time of the synchronization.

BRIEF SUMMARY OF THE INVENTION

The principles of the present invention provide for systems, methods and computer program products for performing synchronization in a flexible manner. Instead of just allowing the user a choice as to what to synchronize and then having the user manually synchronize, one or more of the pair of computer systems involved with the synchronization automatically considers the then-existing circumstances to determine whether, when and how to synchronize. In one described example, the synchronization occurs between a mobile device and a synchronization server. The methods of the present invention may be performed by either synchronizing computer system (e.g., by either the mobile device or the synchronization server).

It is first determined that a data item is to be synchronized. This may be accomplished in response to a user-issued instruction, or in response to a message from the other device indicating that the data item ought to be synchronized. The determination may also have been made by consulting a flexible set of rules that are dictated by a network administrator of the synchronization server as well as by the mobile device user.

Once it has been determined that a data item is to be synchronized, the computer system identifies the available synchronization mechanisms, and then once again consults with the flexible rules to determine which of the available synchronization mechanisms to use. The computer system then synchronizes using the selected synchronization mechanism.

The flexible selection rules may be set and changed by a network administrator and by a user of the mobile device. The flexible selection rules take into consideration the value of the data, the economic cost of synchronization, the security of the synchronization mechanism, and the security of the mobile device. Accordingly, it is much less likely that the flexible rules will allow for highly sensitive data to be shared with an insecure device or over an insecure synchronization channel. Likewise, it is much less likely that a less valuable item of data will be synchronized over an expensive network.

The synchronization rules also take into consideration the location of the mobile device and user. If none of the available synchronization mechanisms are selectable based on the flexible rules, then the synchronization may be delayed, and then the synchronization mechanism may be repeatedly reevaluated until there is a selectable synchronization mechanism.

Should synchronization occur, and later a security condition for synchronization changes, then the effects of the synchronization may be reversed. For example, suppose that the flexible rules indicate that synchronization to a mobile device is proper since the mobile device has configuration settings in which a screen saver is activated after five minutes of non-use, and that a user password is required to restore the device each time the screen saver is activated. Later, suppose the user deactivates the screen saver. The synchronization server may query the mobile device to determine that the screen saver has been deactivated. Since the original conditions for synchronization are no longer met, the synchronization server may issue an instruction to erase the data item from the mobile device, or (if change tracking is enabled) then may instruct the mobile device to roll back the data item to its pre-synchronization state.

Accordingly, the principles of the present invention perform synchronization in a much more appropriate manner considering the surrounding circumstances. In addition, the effects of synchronization may be reversed if security changes so require. Furthermore, the network administrator (and the mobile user) has much more control over when synchronization occurs. This results in a more secure, cost effective, and efficient synchronization design.

Additional features and advantages of the invention will be set forth in the description that follows, and in part will be obvious from the description, or may be learned by the practice of the invention. The features and advantages of the invention may be realized and obtained by means of the instruments and combinations particularly pointed out in the appended claims. These and other features of the present invention will become more fully apparent from the following description and appended claims, or may be learned by the practice of the invention as set forth hereinafter.

BRIEF DESCRIPTION OF THE DRAWINGS

In order to describe the manner in which the above-recited and other advantages and features of the invention can be obtained, a more particular description of the invention briefly described above will be rendered by reference to specific embodiments thereof which are illustrated in the appended drawings. Understanding that these drawings depict only typical embodiments of the invention and are not therefore to be considered to be limiting of its scope, the invention will be described and explained with additional specificity and detail through the use of the accompanying drawings in which:

FIG. 1 illustrates an example of a telephonic device that may implement the principles of the present invention.

FIG. 2 illustrates an example network environment that provides a suitable operating environment for the present invention.

FIG. 3 illustrates a flowchart of a method for flexibly synchronizing in accordance with the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention extends to systems, methods and computer program products for performing synchronization in a flexible manner. Two computer systems in a network each have a local store that contains a copy of a data item that is to be synchronized between the two computer systems. One of the computer systems may be, for example, a mobile device while the other may be a synchronization server. In order to determine whether to synchronize a data item, and what synchronization mechanism to use, one of the computer systems references a flexible set of rules that may be influenced by instructions from a network administrator or a mobile device user.

The flexible set of rules takes into consideration the value of the data, the cost associated with synchronization, the security of the synchronization mechanisms, the security of the mobile device, and/or the location of the mobile device and user in dictating whether and how to synchronize. If synchronization occurs, and the security conditions for synchronization are later not satisfied (e.g., the mobile device's configuration settings have been changed to make the device much less secure), the effects of the synchronization may be reversed. In essence, the principles of the present invention synchronize the appropriate data onto the appropriate device in a manner appropriate to maximize the mobile user's value, benefit, and user experience while at the same time minimize the cost and time of the transaction, and preserving synchronization security.

The embodiments of the present invention may comprise a general-purpose or special-purpose computer system including various computer hardware components, which are discussed in greater detail below. Embodiments within the scope of the present invention also include computer-readable media for carrying or having computer-executable instructions or data structures stored thereon. Such computer-readable media may be any available media, which is accessible by a general-purpose or special-purpose computer system. By way of example, and not limitation, such computer-readable media can comprise physical storage media such as RAM, ROM, EPROM, CD-ROM or other optical disk storage, magnetic disk storage or other magnetic storage devices, or any other media which can be used to carry or store desired program code means in the form of computer-executable instructions or data structures and which may be accessed by a general-purpose or special-purpose computer system.

When information is transferred or provided over a network or another communications connection (either hardwired, wireless, or a combination of hardwired or wireless) to a computer system or computer device, the connection is properly viewed as a computer-readable medium. Thus, any such connection is properly termed a computer-readable medium. Combinations of the above should also be included within the scope of computer-readable media. Computer-executable instructions comprise, for example, instructions and data which cause a general-purpose computer system or special-purpose computer system to perform a certain function or group of functions.

In this description and in the following claims, a "computer system" is defined as one or more software modules, one or

more hardware modules, or combinations thereof, that work together to perform operations on electronic data. For example, the definition of computer system includes the hardware components of a personal computer, as well as software modules, such as the operating system of the personal computer. The physical layout of the modules is not important. A computer system may include one or more computers coupled via a computer network. Likewise, a computer system may include a single physical device (such as a mobile phone, Personal Digital Assistant "PDA", laptop computer, a tablet PC) where internal modules (such as a memory and processor) work together to perform operations on electronic data.

Those skilled in the art will appreciate that the invention may be practiced in network computing environments with many types of computer system configurations, including personal computers, laptop computer, hand-held devices, multi-processor systems, microprocessor-based or programmable consumer electronics, network PCs, minicomputers, mainframe computers, mobile telephones, PDAs, pagers, and the like. The invention may also be practiced in distributed computing environments where local and remote computer systems, which are linked (either by hardwired links, wireless links, or by a combination of hardwired or wireless links) through a communication network, both perform tasks. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

FIG. **1** and the following discussion are intended to provide a brief, general description of a suitable computing environment in which the invention may be implemented. Although not required, the invention will be described in the general context of computer-executable instructions, such as program modules, being executed by computer systems. Generally, program modules include routines, programs, objects, components, data structures, and the like, which perform particular tasks or implement particular abstract data types.

With reference to FIG. **1**, a suitable operating environment for the principles of the invention includes a general-purpose computer system in the form of a mobile device **100**. The mobile device **100** includes a user interface **101** for allowing a user to input information through an input user interface **103**, and to review information presented via an output user interface **102**. For example, the output user interface **102** includes a speaker **104** for presenting audio information to the user, as well as a display **105** for presenting visual information to the user. The mobile device **100** may also have an antenna **109**.

The input user interface **103** may include a microphone **106** for translating audio information into electronic form. In addition, the input user interface **103** includes dialing controls **107** represented by 12 buttons through which a user may enter information. Input user interface **103** also includes navigation control buttons **108** that assist the user in navigating through various entries and options listed on display **105**.

Although user interface **101** has the appearance of a mobile telephone, the unseen features of user interface **101** may allow for complex and flexible general-purpose processing capabilities. For example, mobile device **100** also includes a processor **111** and a memory **112** that are connected to each other and to the user interface **101** via a bus **110**. Memory **112** generally represents a wide variety of volatile and/or non-volatile memories and may include types of memory previously discussed. However, the particular type of memory used in mobile device **100** is not important to the present invention. Mobile device **100** may also include mass storage devices (not shown) similar to those associated with other general-purpose computer systems.

Program code means comprising one or more program modules may be stored in memory **112** or other storage devices as previously mentioned. The one or more program modules may include an operating system **113**, one or more application programs **114**, other program modules **115**, and program data **116**.

While FIG. **1** represents a suitable operating environment for the present invention, the principles of the present invention may be employed in any system that is capable of, with suitable modification if necessary, implementing the principles of the present invention. The environment illustrated in FIG. **1** is illustrative only and by no means represents even a small portion of the wide variety of environments in which the principles of the present invention may be implemented.

FIG. **2** illustrates a network environment **200** in accordance with the present invention. The network environment **200** includes a client computer system **210** (also called herein "client **210**") and a server computer system **220** (also called herein "server **220**") that are "network connectable" to each other over at least a wireless network **230**. In this description and in the claims, "network connectable" means being "network connected" or having the ability to establish a permanent and/or temporary network connection with each other. In this description and in the claims, being "network connected" over a network means having the ability to communicate with each other over at least the network among possibly other networks as well.

The client **210** may be, for example, the mobile device **100** described with respect to FIG. **1**, although the client **210** may be any general purpose and/or special purpose processing system. The client **210** has a local store **211** that may be memory **112** in the case of the mobile device **100**. Local store **211** may hold a variety of different document types including image files, sound files, executable files, word processing document, spreadsheet documents, or the like. For clarity, the local store **211** is illustrated as containing a single data item **212** that is under consideration for synchronization. The data item may be any structured or unstructured data.

The client **210** also includes a networking module **213** configured to send and receive communications over the wireless network **230** to and from the server **220**. Such a networking module may typically be employed within the operating system of the client **210**. A processing module **214** of the client **210** is configured to coordinate access to the data item **212** from the local store, and to use of the networking module **213** so as to perform client operations in accordance with the principles of the present invention.

The server **220** may be, for example, a computer system that runs a synchronization server that synchronizes with a number of wireless devices over the wireless network **230**. However, the server **220** may be any general purpose and/or special purpose processing system. The server **220** also has a local store **221** that stores a data item **222**. The data item **222** at the server **220** and the data item **212** at the client **210** are versions of the same data item. At synchronization, both of these data items are identical. However, between synchronizations, changes may be made to one version, without them being immediately propagated to the other version.

The server **220** also includes a networking module **223** configured to send and receive communications over the network **230** to and from the client **210** as well as potentially other clients that are not shown for clarity. A processing module **224** of the server **220** is configured to coordinate access to the local store **221**, and to use the networking module **223** so as to perform server operations in accordance with the principles of the present invention.

In one embodiment, the client **210** is a mobile device while the server **220** is a synchronization server that provides synchronization services to the mobile device. Although the principles of the present invention are not limited to this embodiment, the following will frequently refer to the embodiment in which the client **210** is a mobile device and the server **220** is a synchronization server. The synchronization server may be a service offered within a corporate network or other common sphere of trust protected by a firewall other security mechanisms from the general public. The mobile device may sometimes be within the corporate network, and may other times be far outside of the corporate network. However, there is no requirement that the synchronization server be within a corporate network. The synchronization server may be, for example, implemented as an Internet service.

FIG. **3** illustrates a flowchart of a method **300** for performing synchronization in a flexible manner considering the then-existing circumstances. The method determines whether, how, and when to synchronize a particular data item by consulting a flexible set of rules that may be set by a user of one of the computer systems, and possible overwritten as dictated by a network administrator. It should be noted that the method **300** may be performed by either the client **210** (e.g., a mobile device) or the server **220** (e.g., a synchronization server).

The flexible set of rules balance the economic value of the data with the economic cost associated with synchronization. The rules also consider the security of the synchronization mechanism as well as the security of the computer system with which synchronization is desired. The rules balance the security risk associated with synchronization with the value in having access to synchronized data. The security rules may also consider the location of the mobile user and/or device in determining whether to synchronize.

The value of the data may be determined by user preferences and/or by a network administrator. For example, to a user, spam e-mails will typically be less important to synchronize. However, e-mails from particular senders (e.g., clients, bosses, business partners, spouses, or the like) may be of greater value to a user. E-mails that contain the words "coin" or "penny" may be of higher value to a penny collector than e-mails that do not contain such words. To a network administrator, contacts stored in a business folder may be more important than contacts stored in a personal folder. While these are just examples, the examples demonstrate that there is a wide variety of criteria from which a value of a data item may be derived based on user preferences and network administrator dictates.

Referring back to FIG. **3**, the method **300** begins by consulting the set of flexible selection rules (act **301**). Based on this consultation, it is determined whether or not to synchronize the data item (decision block **302**). For example, the network administrator or mobile device user may have determined that data items must have at least a predetermined value in order to be synchronized. Spam, for example, might fall below that threshold. If the data item is not to be synchronized (NO in decision block **302**), then the method **300** simply ends without synchronization.

If the data item is to be synchronized based on the consultation with the flexible set of rules (YES in decision block **302**), then the method proceeds to a step for flexibly using an appropriate synchronization mechanism considering the then-existing circumstances (step **303**). This may include any corresponding acts for accomplishing this functional result-oriented step. However, in the illustrated embodiment, the step **303** includes corresponding acts **304**, **305**, **306** and **307**.

Note that in the illustrated embodiment, whether to synchronize is determined by consultation with a set of rules. This determination may also be made by receiving an instruction from the synchronization server, or by receiving a user-issued instruction from the mobile device user. There is no requirement that the flexible set of rules be consulted as to whether the data item is to be synchronized.

Once it has been determined that a data item is to be synchronized, the method identifies which synchronization mechanisms of a number of synchronization mechanisms are currently available (act **304**). For example, a mobile device may be capable of synchronizing with a synchronization server using a number of synchronization mechanisms such as GSM, GPRS, Bluetooth, 802.11a, 802.11b (WiFi), or the like. However, it may be that the mobile device in not within an 802.11a or 802.11b hot spot, or within a Bluetooth network range. Accordingly, the only available synchronization mechanisms would be GSM and GPRS in this example. Note that although the determination of available synchronization mechanisms (act **304**) is illustrated as occurring after the determination as to whether to synchronize the data item, the identification of the available synchronization mechanisms may occur before determining that any particular data item is to be synchronized. For example, the mobile device or the synchronization server may constantly or periodically recevaluate the available synchronization mechanisms. Whether synchronization mechanisms are available may even be a consistently updated configuration setting.

The method also consults the set of flexible selection rules (act **305**) to select one of the synchronization mechanisms (act **306**). For example, the flexible selection rules may not only be used to identify the value of the data, but may also be used to determine the economic costs associated synchronizing the data item over each synchronization mechanism. For example, analog dialup over a Public Switched Telephone Network (PSTN) network or GSM is a relatively slow way of synchronizing, but is currently typically less expensive than synchronization over a GPRS network. If the data item was relatively small, synchronization may be more appropriate over a GSM network since it might not take long even over a lower bandwidth connection to synchronize the data item.

Synchronization costs may also depend on the time of day, or the day of the week. Accordingly, the flexible rules may also consider the time of day, and day of the week in selecting an appropriate network. If the costs associated with a particular synchronization mechanism are to change shortly, then the synchronization may be delayed a short while so as to fall into a less expensive cost structure.

The flexible selection rules also may consider the security of the network associated with the synchronization mechanisms. Some networks are inherently more secure than others. For example, the mobile device may be within the same trusted corporate network as the synchronization server. If remotely connected, the security of the connection (e.g., Virtual Private Network "VPN", Short Message Service "SMS", Secure Sockets Layer "SSL", or direct Remote Access Services "RAS") may be evaluated to determine how difficult it would be for eavesdropping or redirection of the connection.

The flexible selection rules may also consider the security of the mobile device with which synchronization may occur. The mobile device may have some rigid security attributes inherent in the device and in the associated operating system itself, as well as flexible security attributes that may be changed by a user. For example, a WINDOWS XP® enabled laptop computer is generally considered more secure than most Personal Digital Assistants (PDAs). A network administrator may thus dictate to the flexible selection rules that

synchronization of confidential data items may only be made to mobile laptops and not to PDAs.

The flexible selection rules may also consider where the mobile device or user is in determining a synchronization mechanism. For example, if a user is outside of a corporate network, but heading towards an 802.11b hot spot with a secure high-speed VPN connection to the corporate network, the synchronization may be delayed until the user enters the hot spot, wherein synchronization may occur with great speed and security and with very little cost.

Also, suppose the user is working on their local computer within the corporate network. Now suppose the mobile device is, by chance, not connected to the corporate network but is with the user. The synchronization server may send a notification to the user's local computer that there is important information to synchronize. The user may then connect the device to the low cost/high speed corporate network and then synchronize.

Once an appropriate synchronization mechanism is selected (act **306**), then the selected synchronization mechanism is used to synchronize the data item (act **307**). Once synchronized, the synchronization server may constantly or periodically reevaluate whether any security conditions for synchronization have changed such that synchronization is no longer appropriate. For example, suppose that the flexible rules indicate that synchronization to a mobile device is proper since the mobile device has configuration settings in which a screen saver is activated after five minutes of non-use, and that a user password is required to restore the device each time the screen saver is activated. Later, suppose the user deactivates the screen saver. The synchronization server may query the mobile device to determine that the screen saver has been deactivated. Since the original security conditions for synchronization are no longer met, the synchronization server may issue an instruction to erase the data item from the mobile device, or (if change tracking is enabled) may instruct the mobile device to roll back the data item to its pre-synchronization state. Alternatively, the synchronization server may also be setup to change the devices screen saver setting back to the secure mode.

The synchronization may occur automatically based on the consultation with the set of flexible selection rules. Alternatively or in addition, the synchronization server may notify the mobile device user that there is an important item to synchronize and then prompt the user as to whether to synchronize. The user of the mobile device may also indicate that synchronization is desired, and then the synchronization server might then determine which data items are appropriate to synchronize. The user may also manually synchronize by selecting particular data items to synchronize.

Some of the advantages of the present invention may be illustrated by the following scenario involving a fictional stock broker named "Michelle". Having access to up-to-date information is very important to Michelle. The information that she is interested is located in her corporate network in various server locations. She has setup her synchronization server to gather the appropriate data from the appropriate corporate servers and to update her device every morning at 5:00 AM when the data rates are lower with important information that she needs to review before she arrives at work. To do this, she may have created a "weekday morning" sync group that contains: Inbox, Tasks, Calendar, Contacts, high priority action items from her Line-Of-Business (LOB) server, and articles from the Wall Street Journal. This allows Michelle to review important information and plan her day while she eats breakfast and rides the train to work.

When Michelle gets to work, she tethers her mobile phone to her computer and the device does a complete sync. For example, she created a "in-the-office" sync group that contains: Inbox, Tasks, Calendar, Contacts, important information about her customers from the company's LOB server, the latest research information from the company's trading system, and the updates her Wall Street Journal cache. Some of the data items in the "in-the-office" sync group are highly confidential business-related items. The office's network administrator has set a global rule that indicates that such data items should not be synchronized to a mobile phone, but may only be synchronized with laptop or desktop computers with an encrypted file system. Accordingly, although most of the data items in the "in-the-office" sync group are synchronized with Michelle's mobile phone, the highly confidential data items are not. This balances Michelle's need to access information with the office's need to secure confidential information.

Michelle undocks her phone and proceeds to do various work items during the morning. She is scheduled to meet an important customer named "Tom" for lunch. Her Personal Information Management (PIM) server, her sync server and the LOB server are offering her an integrated solution. When her meeting reminder pops up on her PC it also informs her that Tom has some account activity that morning and that she should dock her phone to get this new activity synced to her phone. Michelle docks her phone and Tom's information is updated. During the cab ride, Michelle reviews Tom's account information and activity. During lunch, Tom is impressed how knowledgeable Michelle is about his account and activity.

On her way home from work, Michelle decides she would like to go out to dinner and a movie. On her device, she selects the option to sync the group "weekend personal items" and the individual item restaurant reviews. Her device then syncs movie times, her personal calendar, e-mail, and the weekend weather and restaurant reviews. She finds a movie that she would like to see near her office, finds a review of a new restaurant near the movie. She then calls a friend and makes a date.

Accordingly, the principles of the present invention allow for an intelligence determination of whether, when and how to synchronize by consulting a flexible set of rules that are derived from input from a network administrator as well as the user. The flexible set of rules takes into consideration the value of the data, the economic cost of synchronization, and/or the security of the synchronization mechanisms and device to which synchronization is proposed. Accordingly, synchronization is made in a more intelligent manner.

The present invention may be embodied in other specific forms without departing from its spirit or essential characteristics. The described embodiments are to be considered in all respects only as illustrative and not restrictive. The scope of the invention is, therefore, indicated by the appended claims rather than by the foregoing description. All changes, which come within the meaning and range of equivalency of the claims, are to be embraced within their scope.

What is claimed is:

1. A mobile computing device comprising the following:

a data store;

a networking module; and

a processing module configured to access the data store of the mobile device as well as communicate with the synchronization server over the network using the networking module of the mobile device, wherein the processing device of the mobile device is configured to perform the following:

determine that a data item is to be synchronized;

identify which of a plurality of synchronization mechanisms, including one or more hardwired or wireless communication connections, are available to use for synchronization;

consult a set of one or more flexible selection rules to select a synchronization mechanism, the set of one or more flexible rules taking into consideration value, from having access to synchronized data, relative to at least one of (i) an economic cost for synchronization using each available synchronization mechanism, (ii) network security for each available synchronization mechanism, (iii) security of a computer system, or (iv) value of data being synchronized, and thereby select an available synchronization mechanism appropriate for the data item given the one or more flexible selection rules; and

use the selected synchronization mechanism to synchronize the data item.

2. A first computer system in a network that includes the first computer system having a first data store and second computer system having a second data store, the first computer system comprising one or more computer-readable media having computer-executable instructions for implementing a method for synchronizing the first and second data stores in a flexible manner considering the circumstances that exist at the time of synchronization, wherein the method comprises:

an act of the first computer system determining that a data item is to be synchronized;

an act of the first computer system identifying which of a plurality of synchronization mechanisms, including one or more hardwired or wireless communication connections, are available to use for synchronization;

an act of the first computer system consulting a set of one or more flexible selection rules to select a synchronization mechanism, the set of one or more flexible rules taking into consideration value, from having access to synchronized data, relative to at least one of (i) an economic cost for synchronization using each available synchronization mechanism, (ii) network security for each available synchronization mechanism, (iii) security of the second computer system, or (iv) value of data being synchronized and thereby selecting an available synchronization mechanism appropriate for the data item given the one or more flexible selection rules; and

an act of the first computer system using the selected synchronization mechanism to synchronize the data item with the second computer.

3. A computer system in accordance with claim 2, wherein the first computer system is a synchronization server, and the second computer system is a mobile device.

4. A computer system in accordance with claim 2, wherein the first computer system is a mobile device, and the second computer system is a synchronization server.

5. A computer system in accordance with claim 4, wherein the act of the first computer system determining that a data item is to be synchronized comprises the following:

an act of the mobile device determining on its own that the data item is to be synchronized.

6. A computer system in accordance with claim 4, wherein the act of the first computer system determining that a data item is to be synchronized comprises the following:

an act of the mobile device receiving a user-issued instruction to synchronize the data item.

7. A computer system in accordance with claim 4, wherein the act of the first computer system determining that a data item is to be synchronized comprises the following:

an act of the mobile device receiving a signal from the synchronization server that represents to the mobile device that the data item is to be synchronized.

8. A computer system in accordance with claim 2, wherein the plurality of synchronization mechanisms comprises at least one wireless synchronization mechanism.

9. A computer system in accordance with claim 2, wherein the plurality of synchronization mechanisms comprises a Virtual Private Network (VPN).

10. A computer system in accordance with claim 2, wherein the method further comprises the following:

an act of receiving instructions to change the set of flexible selection rules; and

an act of changing the set of selection rules in response to the instruction.

11. A first computer system in a network that includes the first computer system having a first data store and second computer system having a second data store, the first computer system comprising one or more computer-readable media having computer-executable instructions for implementing a method for synchronizing the first and second data stores in a flexible manner considering the circumstances that exist at the time of synchronization, wherein the method comprises the following:

an act of the first computer system determining whether to synchronize a data item by consulting a set of one or more flexible selection rules, the set of one or more flexible rules taking into consideration value, from having access to synchronized data, relative to at least one of (i) an economic cost for synchronization using each available synchronization mechanism, (ii) network security for each available synchronization mechanism, (iii) security of the second computer system, or (iv) value of data being synchronized, and thereby also determining an available synchronization mechanism appropriate for the data item given the one or more flexible selection rules; and

an act of the first computer system synchronizing the data item with the second computer if the first computer system determines that the data item is to be synchronized based on the one or more flexible selection rules and each available synchronization mechanism, including one or more hardwired or wireless communication connections.

12. A computer system in accordance with claim 11, wherein the first computer system is a synchronization server, and the second computer system is a mobile device.

13. A computer system in accordance with claim 11, wherein the first computer system is a mobile device, and the second computer system is a synchronization server.

14. A computer system in accordance with claim 11, wherein the method further comprises the following:

an act of receiving instructions to change the set of flexible selection rules; and

an act of changing the set of flexible selection rules in response to the instruction.

15. A computer system in accordance with claim 14, wherein the act of receiving instructions to change the set of flexible selection rules comprises the following:

an act of receiving instructions to change the set of flexible selection rules from a user of the first computer system.

16. A computer system in accordance with claim 14, wherein the act of receiving instructions to change the set of flexible selection rules comprises the following:

an act of receiving instructions to change the set of flexible selection rules from an agent of the second computer system.

17. A computer system in accordance with claim **16**, wherein the act of receiving instructions to change the set of flexible selection rules from an agent of the second computer system comprises the following:

an act of receiving instructions to change the set of flexible selection rules from a network administrator of trusted network that includes the second computer system.

18. A computer system in accordance with claim **17**, wherein the method further comprises the following:

an act of receiving instructions to change the set of flexible selection rules from a user.

19. A computer system in accordance with claim **18**, wherein the act of changing the set of flexible selection rules in response to the instruction, comprises the following:

an act of fulfilling the instructions received from the network administrator of the second computer system to the extent that there is a conflict between the instructions received from the network administrator of the second computer system and the instructions received from the user of the first computer system.

20. A computer system in accordance with claim **11**, wherein the method further comprises the following:

after using the selected synchronization mechanism to synchronize the data item, an act of determining that the conditions for synchronization are no longer met in light of the flexible selection rules; and

an act of automatically reversing the synchronization if it has been determined that the conditions for synchronization are no longer met.

* * * * *