(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
30 May 2003 (30.05.2003)

PCT

(10) International Publication Number
WO 03/044662 A1

(51) International Patent Classification^7: G06F 9/445

(21) International Application Number: PCT/US01/42947

(22) International Filing Date: 15 November 2001 (15.11.2001)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant *(for all designated States except US)*: **ALADDIN KNOWLEDGE SYSTEMS, LTD.** [IL/IL]; Beit Oved Street 15, 67021 Tel Aviv (IL).

(72) Inventor; and
(75) Inventor/Applicant *(for US only)*: **OS, Ron, Van** [—/US]; 715 Timberpine Avenue, Sunnyvale, CA 94086 (US).

(74) Agent: **CASCIO, Anthony, T.**; Burns, Doane, Swecker & Mathis, LLP, P.O Box 1404, Alexandria, VA 22313 (US).

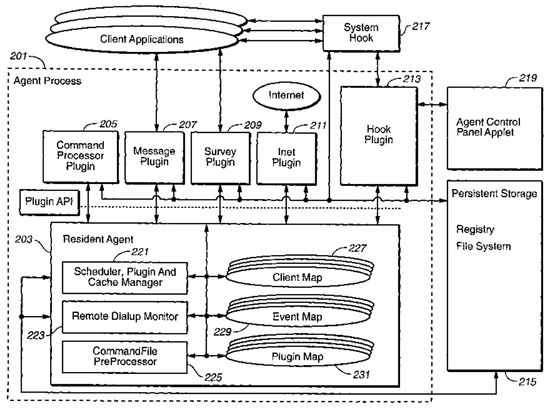
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: INCREMENTALLY INCREASING OR DECREASING THE AVAILABLE FUNCTIONALITIES OF A COMPUTER PROGRAM



(57) Abstract: The present invention, generally speaking, allows for incremental distribution of a computer program through a two-way automated exchange of information between a user's machine and a server machine via a wide area computer network, e.g., the Internet. The user initially purchases the core program. The core program may be distributed electronically. The menu structure of the core program may include menu items relating to capabilities that are not part of the core program. When the user selects menu items not directly supported by the core program, a dialog is displayed asking the user whether the user wishes to download a program module corresponding to the selected menu item, either immediately or in the background at the next available opportunity. The program module may be distributed on either a "Buy/Try" or "Try/Buy" basis, Try/Buy being preferred such that the user is afforded an opportunity to use the new program feature for a period of time before committing to buy the additional program module. The menu structure of the program may also be dynamically updated to include menu items relating to capabilities developed after distribution of the core program. Program modules implementing these capabilities may be downloaded and purchased in the same manner.

WO 03/044662 A1

INCREMENTALLY INCREASING OR DECREASING THE AVAILABLE FUNCTIONALITIES OF A COMPU TER PROGRAM

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates to software distribution and particularly to electronic software distribution (ESD).

State of the Art

Application software has become increasingly sophisticated and feature-laden. Many application programs are now aptly described by the proverbial 80/20 rule—80% of program usage is confined to 20% of the program features; the remaining 80% of program features account for only about 20% of program use. Custom installation capabilities have been developed in view of this phenomenon. That is, the user may choose to install only a subset of the program that the user expects to actually use, saving disk space on the user's machine. Nevertheless, the user is generally required to purchase the entire application program at one time. To draw an analogy, the user is required to purchase an entire seven-course meal when all the user may really have wanted is a good dessert.

As electronic software distribution becomes increasingly prevalent, the largely monolithic nature of existing software becomes an increasingly greater inconvenience. Large programs take a long time to download, and long downloads are easily interrupted, causing them to fail. Even where multiple partial downloads are possible, the user is noticeably inconvenienced.

The monolithic nature of existing software programs also makes upgrade problematic. Upgrade typically requires a significant investment and entails significant effort and inconvenience. Again, the transfer of a large binary image is required.

On the software publisher's side of the equation, large, monolithic, expensive software programs limit software publisher's sales strategies and market penetration. Distribution costs remain unnecessarily high.

SUMMARY OF THE INVENTION

The present invention, generally speaking, allows for incremental distribution of a computer program through a two-way automated exchange of information between a user's machine and a server machine via a wide area computer network, e.g., the Internet. The user initially purchases the core program. The core program may be distributed electronically. The menu structure of the core program may include menu items relating to capabilities that are not part of the core program. When the user selects menu items not directly supported by the core program, a dialog is displayed asking the user whether the user wishes to download a program module corresponding to the selected menu item, either immediately or in the background at the next available opportunity. The program module may be distributed on either a "Buy/Try" or "Try/Buy" basis, Try/Buy being preferred such that the user is afforded an opportunity to use the new program feature for a period of time before committing to buy the additional program module. The menu structure of the program may also be dynamically updated to include menu items relating to capabilities developed after distribution of the core program. Program modules implementing these capabilities may be downloaded and purchased in the same manner.

The functionality of a program can not only be increased but might also be decreased incrementally. In particular, a software publisher may be afforded the ability to remotely control the availability of one or more features of an application. Enablement/disablement is achieved by intercepting key operating system messages and only permitting authorized activities to complete within the target application. No source code changes are required. Furthermore, the Internet may be used for remote control of the application.

The foregoing capabilities may be implemented by a software agent installed on the user's machine. The agent may be distributed and installed together with the core program, for example. Except as the user may direct, communication between the agent and the server is optimized to be unobtrusive or transparent, using spare bandwidth of intermittent Internet connections, for example. The agent is software non-specific and may be instructed to operate with respect to any arbitrary software program, and may further be instructed at various times to operate with respect to various different software programs, including multiple different software programs on a single machine.

BRIEF DESCRIPTION OF THE DRAWING

The present invention may be further understood from the following description in conjunction with the appended drawing. In the drawing:

Figure 1 is a generalized block diagram of a system with which the present invention may be used;

Figure 2 is a more detailed block diagram of the agent of Figure 1;

Figure 3 is a diagram illustrating a distributed server architecture that may be used in the system;

Figure 4 is an illustration of a screen display in which the time and place of message delivery is controlled;

Figure 5 is a flowchart showing a general sequence of execution events;

Figure 6 is an illustration of a screen display showing an example pull-down menu;

Figure 7 is an illustration of a screen display showing the pull-down menu of Figure 6 modified to disable a menu item;

Figure 8 is an illustration of a screen display showing an Enhanced Feature dialog displayed upon selection of an added menu item within the pull-down menu of Figure 6;

Figure 9 is an illustration of a screen display showing another example pull-down menu;

Figure 10 is an illustration of a screen display showing the pull-down menu of Figure 9 modified to add menu items.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

A persistent client architecture that may be used to implement incremental software distribution will first be described, followed by a description of the application of the persistent client architecture to incremental software distribution.

Referring now to Figure 1, the general architecture of the present system will be described. A user machine 101 is assumed to include a run-time environment 103, provided by an operating system, a Web browser or the like, and to be running one or more computer programs 105. The computer programs may be software applications, system software or embedded computer programs. Individualized, interactive program execution flow is made possible by equipping the user machine with a persistent client, or persistent agent 107, that engages in two-way communication with a server 109. The agent may be installed concurrently with an application, may be pre-loaded on the user's machine, may be separately installed from a disk or download, etc. The connection between the agent and the server may be a virtual connection, i.e., a connection that time-shares a physical communications channel with other communications. In an exemplary embodiment, the virtual connection uses spare bandwidth of Internet connections, continuous or intermittent, to communicate with the server. The agent can and typically does interact with the application without a concurrent Internet connection.

Communications between the agent and the server are preferably two-way. In the uplink direction, the agent communicates control, configuration and usage information (and in some embodiments, registration information, survey information, etc.). In the downlink direction, the server communicates non-executable content, executable content, or both, including control information, agent updates, etc. Separate servers may be responsible for delivering non-executable content and executable content, as described more fully hereinafter. Note that non-executable content may nevertheless be active, i.e., contain HTTP links enabling the user to "click through" to related Web sites. Executable content may relate to the application or to the agent or both. Executable content related to the application may include updates, bug fixes, additional code modules, etc. Executable content related to the agent allows the agent to be transparently upgraded with new capabilities in the field, avoiding the potential problem of agent obsolescence.

Referring to Figure 2, a detailed block diagram of the agent is shown.

In an exemplary embodiment, the agent follows a plug-in architecture. An agent process 201 therefore includes a resident agent 203 and various plug-ins that interface to the resident agent through a plug-in API. In an exemplary embodiment, the plug-ins include a command processor plug-in 205, a message plug-in 207, a survey plug-in 209, an Inet plug-in 211 that handles virtual connections to the Internet, and a hook plug-in 213. The message plug-in, survey plug-in, and possibly other plug-ins are capable of taking actions within the process and User Interface (UI) space of the client applications. Other plug-ins may be included with the agent or added to the agent by download. If a plug-in needs the assistance of another plug-in, the agent will pass parameters transparently to the target plug-in. Persistent storage 215 is provided for the plug-ins as well as for the resident agent, e.g., within the registry file system.

The modularity resulting from plug-in architecture of the agent is important from the standpoint of allowing for user-transparent operation. The core agent and the plug-ins are all small modules that are easily downloadable. The time needed to download a module is typically only a few seconds.

Core tasks of the agent include the following:

1. Manage plug-ins and inter-plug-in communication.
2. Download content (command files) and determine an appropriate command interpreter for handling the command files. Retrieve the command interpreter plug-in from the server and invoke it with the downloaded command file.
3. Maintain state (e.g., the current command file) to survive system crashes and restarts. The operating system registry may be used for persistent storage of state information including the configurations of the plug-ins, the status of events and the registered client applications.
4. Monitor the system Internet connection and schedule uploads and downloads.
5. Track target applications and determine their usage. From this information and the command file data, schedule actions to be taken in the target applications user interface (UI) space through the plug-in interface.
6. Change its level and type of activity, including becoming inactive in response to a server.

The agent is capable of interacting with software applications in all respects without modification of the application itself. In particular, a small system hook

217 (e.g., a linkable code module) is inserted into the message processing loop. Using data provided by the agent, the system hook determines if any relevant actions are happening within a monitored application and if so, passes this information off asynchronously to the hook plug-in 213. The system hook is designed to not degrade the user's system performance or application performance. More particularly, in an exemplary embodiment, the agent when it first launches loads the hook plug-in 213, which starts execution of a separate thread. This thread interacts with the system hook 217 and is responsible for selecting messages of interest. The separate thread ensures that processing of the messages of the client application are not noticeably slowed down. Note that, to prevent recursion in the message processing, the system hook ignores any messages related to the agent itself.

The Inet plug-in 211 is responsible for handling all Internet traffic. In an exemplary embodiment, it supports various types of Internet transactions, including registering an agent with the server and obtaining a user ID, retrieving a command file using the user ID, uploading data to the server, and downloading resources from the server. Data may be exchanged using POST and GET commands, for example, as in the HTTP1.1 or higher protocol. The Inet plug-in is designed to gracefully fail if any transaction is not completed across the Internet.

The command processor plug-in 205 is responsible for converting the command file into tangible actions. For example, it scans the command file and schedules all resource downloads required by the command file, expands any macros, and generates a clean version of the command file. It then processes the command file, merges it with existing command files, removes all completed events from the command file, and schedules all events and actions to be taken by the agent. Finally, it marks the command file as active in persistent storage and uploads a command line status update that allows the server to track the execution of events in the client application.

An agent control panel applet 219 may be provided to enable user interaction with the agent to control prospective operation of the agent, although typically the user will not have occasion to use the control panel. For purposes of the present application, the agent control panel applet is assumed to be inactive.

In an exemplary embodiment, the resident agent includes a scheduler/manager 221, a remote dialup monitor 223, and a command file pre-processor 225. The resident agent also includes a client map 227, an event map 229 and a plug-in map 231. The resident agent is responsible for dynamically maintaining the configuration and status of active plug-ins, the registered client applications and the events working on the client applications. A command queue contains actual event information and is processed upon each agent start. In an exemplary embodiment, the agent is started by a machine start table within the registry of the operating system.

The scheduler/manager 221 is responsible for establishing periodic Internet connections with the server, through the Inet plug-in 211. If a connection becomes available, each client object is allowed bandwidth to service the client's needs. Subsequently, all pending POST operations are processed. The scheduler/manager can be invoked either via an event driven method, in the case of dialup Internet access, or at periodic intervals in the case of direct (or proxied) LAN-based Internet access. In the case of dialup access, different dialup access methods may be used depending on the software configuration of the user machine. The remote dialup monitor 223 determines which dialup access method is used and establishment of an Internet connection is detected accordingly.

The client map 227, event map 229 and plug-in map 231 together operate to establish "client channels" though which interaction between the clients and the server occurs. The client map consists of one or more client objects. At a minimum, a privileged client object is present that is allowed to add clients to and remove agents from the client map and to add agents to and remove agents from the plug-in map. All other client channels can only be used to schedule events and direct the agent to download content from a server. A client object within the client map has a corresponding event object within the event map and a corresponding plug-in object within the plug-in map. The event map in combination with the client map causes user interactions in the client applications UI space.

Note that preferred support for copies of applications already in the field can be added simply by causing the agent to download client objects for those applications. A client object (or "affinity module") contains information that allows the system hook to recognize events from a particular application.

Referring again to Figure 1, the agent 107 checks in with the server 109 when a check-in interval for the application has elapsed. The agent may receive back a command file from the server, which the agent then interprets. The interpretation of the command file may cause the agent to fetch resources from the server and/or place information back onto it. The agent may also be instructed to check-in for another command file. The privileged client is also considered an application for the agent. Therefore the agent checks in with the appropriate server on a check-in interval separate from the check-in intervals of other applications. Also, an application's command file may cause the privileged client to check in, or vice versa.

When the agent 107 has acquired the resources and commands from the server 109 to actually do some work, it can be instructed to immediately display appropriate messages to the user, or (more commonly) to wait until the target application is running, and work in the context of the application. The agent converts system event data into tangible actions events for the attached plug-ins, with messages appearing to the user as coming from the vendor, within the application's screen window and only while the application is running. Referring again to Figure 2, a typical sequence of events is as follows:

1. The system hook 217 determines that a new application has launched or obtained the user's focus.
2. The resident agent 203 queries its client objects to see if the application is a client. If it is not a client, the agent remains dormant.
3. A valid client with user input will cause the resident agent 203 to instruct the system hook 217 to start detailed monitoring of the application and route selected application messages through the hook plug-in 213.
4. The hook plug-in 213 will reflect the message asynchronously to the resident agent 203, which will catalog the events under the current user's name.
5. The resident agent 203 queries its client/event map 227, 229 to look for a match.
6. If a match exists, the event is executed, which could include invoking a plug-in to undertake action in the application's UI space. If visible content is shown in the application's UI space, the client application is temporarily disabled and cannot receive user focus.
7. If any uploadable content is generated during this event, it is passed to the Inet plug-in 211, which will either send it or schedule it to be sent the next time bandwidth is available.
8. After completion of the event, the user focus is set back to the client application.
9. The agent returns to Step 4 above until the client application loses focus. When the client application loses focus, the agent transfers any client application-related data to persistent storage, at which point the agent reverts to Step 1 above.

If the agent 107 were to somehow be disabled and hence unable to intercept UI messages, then the user would be able to access functionality intended to be disabled and would be unable to access added functionality. To ensure that the agent 107 remains operational during the time that a target program is run, the program can be "injected" with or otherwise have added to it a code stub in a manner described, for example, in the previously-referenced U.S. patent application. A call is made by the code stub from within the injected software program to the agent to verify that the agent is in fact operation before the user is allowed to begin using the software program or before all of the functionality of the application will be unlocked and made available to the end user.

Various alternatives allow for such code injection to be performed. In one alternative, the application components are injected at the time that the final application installer is built ("early binding"). In accordance with another alternative, the application installer is combined with the agent installer which, during installation on the end user machine, will inject code into the appropriate application modules to ensure that the agent is running.

Having described the structure and function of the agent, the server will now be described.

Referring once again to Figure 1, the essential job of the server 109 is the delivery of an appropriate command file to particular agent. The command files in the agent determine the action that the agent is going to take—which of the various kinds of activities it will carry out, at what time, with respect to what user operation, etc. The server maintains a record for every single user of an application. When the agent working for one of its user's connects to the server, it consults a table of rules that determines which, if any, of the potential command files that the server has for that application are appropriate for that agent. Those rules are predicates that are based on all the data in a database 111 relative to that user.

An example of a rule might be "If installation of this application took place 60 or more days ago, send Command File A," which causes the agent to perform some action, "and if installation took place less than 60 days ago, send Command File B," which takes some other action. The two actions would differ with respect to the degree of experience that particular user has with the program. For example, in the case where an upgrade has become available, a publisher may choose to send one upgrade message to experienced users, more appropriate to their experience level, and another upgrade message to less experienced users, more appropriate to their experience level. The determination of experience level may be based, for example, on the time elapsed since installation.

The server uses a rules engine 111 to apply rules that have been created in a table sequentially to determine which if any of those rules are true for a particular agent that is querying the server at a particular point in time. Upon discovering that one or more of those rules "fires," i.e. is true, then the corresponding one or more command files are downloaded to the agent. The publisher therefore enjoys very "fine-grain" control of the activities of an agent based on the attributes of that agent. Very sharp targeting results in which particular information is sent to particular agent based its characteristics and its history.

As may be appreciated from the foregoing description, the server has two different types of responsibilities. One function of the server is to maintain the agent. Operations to maintain the agent occur through the control channel described previously. Another function of the server is to provide customer support for specific client applications. These two distinct functions can be combined on one physical server machine or on multiple physical server machines. More preferably, these two functions are separated, with agent maintenance being handled by a technology provider server and customer support being handled by a collection of software vendor servers. In general, executable content is provided from the technology provider server 301 across the control channel and non-executable content is exchanged with software vendor servers 303a - 303n across other channels as shown in Figure 3. In this manner, executable content may be assured to be virus-free. Also, private vendor-customer information may be passed directly to the vendor without being passed through a third party. For tracing and billing purposes, the privileged client periodically connects to the technology provider server and informs it of activities of the agent on behalf of various client applications.

Identifiers are allocated to support the foregoing separation of functions. In particular, the agent when it is first activated seeks a connection opportunity and, when a connection is established, obtains an agent ID 305 from the technology provider server through the control channel. At the same time or at a later time, the agent receives from the server a command file instructing the agent to look for a particular application. If that application is found installed on the user's system, a client ID 307 is obtained for that copy of the application. Only the technology provider server need be aware of the correspondence between agent IDs and client IDs. Transactions between the agent and the vendor server use only the client ID.

At a vendor server, a Database Management System (DBMS, 309a - 309n) maintains a per-client-copy database of information uploaded from various instances of an application. In an exemplary embodiment, the agent collects numerical counts for each menu bar item in a client application. The vendor may determine from the database how often the file:print command has been used, for example. The DBMS includes a rules engine. Business rules are established governing the actions to be taken in relation to a particular copy of the application depending upon the data stored for that application. When action is to be taken, a command file is prepared and transferred to the agent.

Note that the system has the ability to precisely target the moment a message dialog appears in a client application. The vendor can pick an operation from among a menu hierarchy of the application, a time delay, and a number of times to repeat the operation until it is completed with a click-through or other affirmative response. The system also has the ability to determine who among the vendors installed base will see a particular message. Criteria can be based on demographics, responses to past offers, responses to past surveys, usage information, time since the application was installed, even random selection. Any information in the database can be used to determine who gets a particular message. For example, a particular rule might send all users a message before or after use of a particular feature after that feature has been used a specified number of times (Figure 4).

Preferably, a Web-based administration tool is provided to allow business rules to be set up and changed through a familiar Web-form interface.

Application of the persistent client to incremental software distribution will now be described. In general, the foregoing agent technology lends itself to working with legacy applications, which have no native mechanism for reducing or apportioning their respective feature sets, nor any mechanism for increasing their feature sets.

The described agent technology is able to attach itself into the main window of the target application. This attachment allows direct access to all of the window objects that the target application generates for constructing the user interface. Some of these elements are the actual menu bar and menu items. These window objects can be dynamically updated by code added at runtime to the target application to accomplish: 1) removal of a menu item constructed by the target application; 2) addition of a menu item and installation of a command handler to add functionality; 3) disabling a menu item, by graying it out; 4) rerouting a menu command handler such that the attached code gets the first chance to handle the user command. The latter feature allows for display to the user of a message informing the user that the feature is for sale. A combination of these capabilities allows the agent to tailor a legacy application for incremental deployment.

Many menu commands are accessible either through a mouse click or a keyboard shortcut. In either event, the OS will formulate the correct windows message to send the menu object to invoke a command. This higher-level message is intercepted to handle the menu object interaction in the manner specified for the particular application.

To reduce the feature set of an application, the agent can either: 1) Remove the menu item from the application at runtime, to make the feature unavailable to the user; or 2) Intercept the user interaction before it reaches the application and substitute a dialog alerting the user of what steps to take to acquire the functionality. The steps could allow for a Try/Buy scenario. In order to positively identify messages to be intercepted, it may be necessary to for the technology provided run a test setup in which a copy of the target application is run. A test person performs each UI action to be disabled while a separate computer program (which may be the agent previously described) monitors and records the corresponding messages.

To increase the feature set of an application, the agent performs the following steps: 1) Insert a menu item into the application at runtime, to make the feature available to the user; and 2) Handle the user interaction with this new menu item and invoke the acquired functionality.

Any operating system (OS) that provides a Graphical User Interface (GUI) handles the keyboard and pointing device input from the user and converts them into a format (messages) suitable for processing by applications using the GUI services. The agent can insert itself between the GUI and the application, thereby monitoring and controlling the user input received by the application. In addition, the agent can generate additional GUI messages to control the UI of the application. As an example, when the OS detects that the user has clicked a mouse button, it will determine which application has the user focus and should receive a message describing this event. Once this message is received by the application, it can execute some feature. Being able to monitor and control the flow of messages between the OS and the application enables the agent to alter the application behavior at runtime. Further details concerning monitoring and control of the flow of messages between the OS and the application may be found in U.S. Application Serial No. 09/138,403 filed August 24, 1998 (Attys. Dkt. No. 031994-026), incorporated herein by reference.

The application UI is constructed by issuing requests to the OS for generating and displaying UI elements on the end user's monitor. Examples are a window with a title bar, a button, an edit text box, and so forth. By issuing requests to the OS, the agent can add or remove UI elements from the application's UI. In the case of the WindowsTM operating system, WIN32 API calls are used for this purpose.

More particularly, during execution, the application constructs memory objects that represent the various objects rendered on the screen. These objects are called windows (window objects), well-known in the art. All of the UI elements are managed by the OS on behalf of the program and become part of the desktop. As all of these objects are part of the public desktop, any other application, for example the agent previously described, can query the system for what is on the desktop. Once an object of interest is found, it can be injected with a small hook code module which will execute in the context of the target application. Once part of the application, it can perform various functions, e.g., adding/removing UI elements (menus), filtering the messages received by the OS, etc.

As an example, an applications menu structure can be altered at runtime by sending the appropriate messages. Once menu items have been added to an application to extend its feature set, the user is able to select them. If the user selects the item, the persistent client can intercept the corresponding message and execute the appropriate code. This code can be part of the original application or new functionality that become available after the application had already been released. In this case, the persistent client can display a dialog asking the user if she or he wants to download the new feature corresponding to the new menu item, either immediately or in the background.

Referring more particularly to Figure 5, an execution example is shown leading to the user being asked to add a new feature to a product. Once the OS has handled a user action, in this case the selection of a menu item, it formats a message describing this event and would normally send it to the application. In the present instance, the message is intercepted by the agent and compared against a list of dynamically added menu items (501). If the message did not originate from a dynamically added menu item, the message is passed to the application and processed as usual (503). If the message did originate from a dynamically added menu item, the persistent client will check to see if the user has already downloaded (or unlocked) the associated feature code (505). If the feature code is in place, the client will execute the code (507). If the feature has not been downloaded yet, a dialog is presented to the user informing the user of the feature availability and steps to acquire it (509).

The program module may be distributed on either a "Buy/Try" or "Try/Buy" basis, Try/Buy being preferred such that the user is afforded an opportunity to use the new program feature for a period of time before committing to buy the additional program module. Try/Buy self-wrapping of software programs is described in U.S. Patent Application Serial No. 08/921,394 entitled MULTI-TIER ELECTRONIC SOFTWARE DISTRIBUTION, filed August 29, 1997, incorporated herein by reference. The menu structure of the program may also be dynamically updated to include menu items relating to capabilities developed after distribution of the core program. Program modules implementing these capabilities may be downloaded and purchased in the same manner.

Referring more particularly to Figure 6, an example is shown of a native target application's SEARCH menu. Using the techniques described, the agent is able to disable a menu item in the target application, as shown in Figure 7. The "Find Next F3" item has been dynamically disabled, preventing the user from selecting this menu item. Alternatively, the menu item selection can be rerouted and handled by the agent.

Instead of executing the corresponding function, a dialog can be shown to allow the user to acquire the requested feature, as shown in Figure 8. If the user decides not to acquire the feature, it can be disabled as in Figure 7.

Steps performed by the agent to allow it to modify the attributes of the menu items of the target application include identifying the target application, injecting a small hook into the message processing loop, and retrieving a handle to the target application frame. A code example illustrating this processing is provided as Appendix A.

Special care needs to be taken to ensure that a disabled menu item is no longer accessible using short cuts. The injected code in the message processing loop can selectively suppress windows messages passed from the operating system to the target application. This mechanism ensures that disabled items are no longer available to the target application. In addition, this mechanism allows the agent to redirect messages intended for the target application to itself and handle the message, e.g., by displaying to the user that the particular item is disabled and can be purchased as a separate module. An example procedure for filtering messages in this manner is provided as Appendix B.

The agent also provides for the dynamic addition of menu items at runtime. For example, the native target application's FILE menu may appear as shown in Figure 9. An example of a corresponding runtime-modified menu is shown in Figure 10, in which two additional items have been added to allow the user to either email or fax the document to a recipient. Steps performed by the agent to allow it to modify the attributes of the menu items of the target application include identifying the target application, injecting a small hook into the message processing loop, and retrieving a handle to the target application frame. A code example illustrating this processing for the addition of menu items is provided as Appendix C.

When the user selects a new menu item, the added command is handled by the injected code which monitors the windows messages received by the target application frame. When a command from a dynamically added menu item is selected the corresponding code can either execute inside the context of the target application or out of process. In the former instance, the injected code handles the message: HandleInProcessMenuCommand (...). In the latter instance, the message is forwarded to the agent using PostMessage (...). A code example illustrating these steps is provided as Appendix D.

APPENDIX A

```
//Dynamically change menu item attributes
//
CWnd      pClientWnd;

if(pClientWnd.Attach(ClientWnd))
{
    CMenu*     pClientTopMenu = pClientWnd.GetMenu();

    if(pClientTopMenu)  //Frame Menu
    {
        CMenu*pClientHelpMenu = pClientTopMenu->GetSubMenu(2);

        if(pClientFileMenu)         //Search menu
        {
          if(Disable)
          {
              //disable menu item
              //
              pClientHelpMenu->EnableMenuItem (m_TargetMenuCmd, MF_DISABLED | MF_BYCOMMAND);
          }
          else
          {
                //enable menu item
                //
              pClientHelpMenu->EnableMenuItem (m_TargetMenuCmd, MF_ENABLED | MF_BYCOMMAND);
          }
        }
    }

    pClientWnd.Detach();
}
```

APPENDIX B

```
LRESULT CALLBACK FilterMsgProc(INT hc, WPARAM wParam, LPARAM lParam)
{
      static      WORD  CurrentMenuCmd = 0;
                  PMSG  pmsg = (PMSG)lParam;

      if (hc == MSGF_MENU)
      {
            if((pmsg->message == WM_MENUSELECT) && (pmsg->lParam != 0x00))
            {
                  if(!(HIWORD(pmsg->wParam) & MF_POPUP) && (LOWORD(pmsg->wParam)))
                  {
                        CurrentMenuCmd = LOWORD(pmsg->wParam);
                  }
                  else
                  {
                        CurrentMenuCmd = 0;
                  }
            }
            else if(CurrentMenuCmd && ((pmsg->message == WM_LBUTTONDOWN) || (pmsg-
            >message == WM_LBUTTONUP)))
            {
                  if(CurrentMenuCmd == m_TargetMenuCmd.cmd)
                  {
                        //First close the Menu in Question and than put up a dialc
                        //
                        MessageBox(pmsg->hwnd, "Has disabled this Cursor accessed
                        Feature", "eBoomerang", MB_ICONSTOP | MB_OK);
                        return 1;
                  }
            }
            else if((pmsg->message == WM_CHAR) || (WM_SYSCHAR  == pmsg->message))
            {
                  if(pmsg->wParam == m_TargetMenuCmd.kbd)
                  {
                        //First close the Menu in Question and than put up a dialc
                        //
                        MessageBox(pmsg->hwnd, "Has disabled this Accelerator
                        Feature", "eBoomerang", MB_ICONSTOP | MB_OK);
                        return 1;
                  }
            }
            else if(CurrentMenuCmd && (pmsg->message == WM_KEYDOWN) && (pmsg->wPa
            == 0x000D))
            {
                  if(CurrentMenuCmd == m_TargetMenuCmd.cmd)
                  {
                        //First close the Menu in Question and than put up a dialc
                        //
                        MessageBox(pmsg->hwnd, "Has disabled this KeyBoard accesse
                        Feature", "eBoomerang", MB_ICONSTOP | MB_OK);
                        return 1;
                  }
            }
      }
      return 0;
}
```

APPENDIX C

```
//Dynamically add or remove menu items
//
CWnd      pClientWnd;

if(pClientWnd.Attach(ClientWnd))
{
    CMenu*      pClientTopMenu = pClientWnd.GetMenu();

    if(pClientTopMenu)  //Frame Menu
    {
        CMenu*pClientHelpMenu = pClientTopMenu->GetSubMenu(0);

        if(pClientFileMenu)        //File menu
          {
          if(Add)
          {
                //Insert at the almost 'Exit' position
                //
                pClientHelpMenu->InsertMenu (m_ExitMenu, MF_BYCOMMAND, m_ExtramenuCmd[0],
                m_ExtraMenuTxt[0]);
                pClientHelpMenu->InsertMenu (, m_ExtramenuCmd[0], MF_BYCOMMAND, m_ExtramenuCmd[1],
                m_ExtraMenuTxt[1]);
                pClientHelpMenu->InsertMenu (m_ExtramenuPos[0], MF_BYPOSITION,  MF_SEPARATOR);
                pClientHelpMenu->InsertMenu (m_ExtramenuPos[1], MF_BYPOSITION,  MF_SEPARATOR);
          }
          else
          {
                  //Remove the menu
                  //

                  pClientHelpMenu->RemoveMenu(m_ExtramenuCmd[0], MF_BYCOMMAND);
                  pClientHelpMenu->RemoveMenu(m_ExtramenuCmd[1], MF_BYCOMMAND);
                  pClientHelpMenu->RemoveMenu(m_ExtramenuPos[0], MF_BYPOSITION);
                  pClientHelpMenu->RemoveMenu(m_ExtramenuPos[1], MF_BYPOSITION);

           }
         }
    }

    pClientWnd.Detach();
 }
```

APPENDIX D

```
BOOL WINAPI HookProc(HWND hwnd, UINT uiMessage, WPARAM wParam, LPARAM lParam)
{
   static   UINT  Lastmessage = WM_HOOKCLIENTDEACTIVATE;
   static   HWND  LasthWnd = NULL;
   static   HWND  Owner = NULL;
   static   DWORD LastPid = 0;
   static   DWORD Command = 0;

   switch(uiMessage)
   {
      case WM_FORWARD_UNWINDING_MSG:
        hhkGetMessage = (HHOOK) lParam;
      break;

      case WM_FORWARD_UNWINDING_WND:
        hhkCallWndProc = (HHOOK) lParam;
      break;

      case WM_COMMAND:

      if ((ghwndInjectRelay == NULL) || !IsWindow(ghwndInjectRelay))
      {
        //Check if eBoomerang Client is still running
        FindInjectRelayWindow();
      }

      if (!hhkGetMessage && (ghwndInjectRelay != NULL))
      {
        //Ask for an upload of the getMessage hook members
        //
        PostMessage(ghwndInjectRelay, WM_FORWARD_UNWINDING_MSG, (WPARAM) hwnd,
        (LPARAM) GetCurrentProcessId());
      }

      if (!hhkCallWndProc && (ghwndInjectRelay != NULL))
      {
        //Ask for an upload of the callWndProc hook members
        //
        PostMessage(ghwndInjectRelay, WM_FORWARD_UNWINDING_WND, (WPARAM) hwnd,
        (LPARAM) GetCurrentProcessId());
```

APPENDIX D (cont.)

```
      }

      if ((ghwndInjectRelay != NULL) && (hwnd != ghwndInjectRelay))
      {
        if(HIWORD(wParam) <= 1) //menu or accelerator
        {
             for(int i=0; i< m_maxcmds; i++)

             //Check for a dynamically added command
             //
             if((m_ExtramenuCmd[i] == LOWORD(wParam))
             {
                if(INPROC)
                       HandleInProcessMenuCommand(m_ExtramenuCmd[i]);
                else
                       PostMessage(ghwndInjectRelay, WM_HOOKCLIENTCOMMAND,
                       (WPARAM) uiMessage, (LPARAM) hwnd);
             }
        }
      }
      return TRUE;
      }
   }

   return FALSE;
}
```

It will be appreciated by those of ordinary skill in the art that the invention can be embodied in other specific forms without departing from the spirit or essential character thereof. The presently disclosed embodiments are therefore considered in all respects to be illustrative and not restrictive. The scope of the invention is indicated by the appended claims rather than the foregoing description, and all changes which come within the meaning and range of equivalents thereof are intended to be embraced therein.

What is claimed is:

1. A method of software distribution, comprising the steps of:

installing on an end user machine a program that displays command menus from which the user selects menu items;

instructing a supervisory agent as to which menu items are to be enabled and which menu items are to be disabled;

when a menu item is selected, the supervisory agent intercepting a corresponding message to the application; and

the supervisory agent passing the message along to the application only if the menu item is enabled.

2. The method of Claim 1, further comprising taking steps to ensure that disabled functionality cannot be restored by disabling the supervisory agent.

3. The method of Claim 1, further comprising blocking substantially all code paths that would invoke disabled functionality.

4. The method of Claim 1, comprising the further step of changing instructions of the supervisory agent as to which menu items are to be enabled and which menu items are to be disabled.

5. The method of Claim 4, wherein the instructions are changed from a remote computer.

6. The method of Claim 5, wherein the instructions are changed over the Internet.

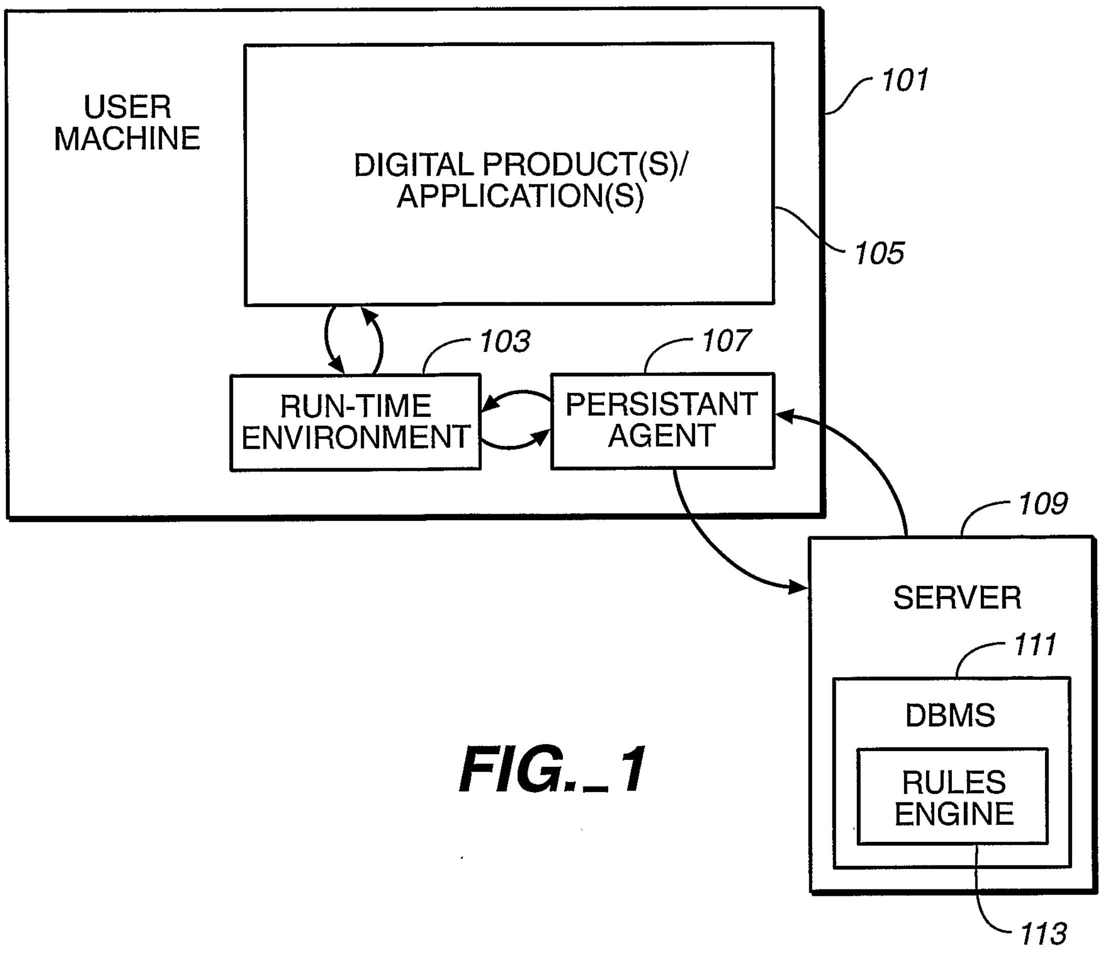

USER MACHINE
DIGITAL PRODUCT(S)/ APPLICATION(S)
RUN-TIME ENVIRONMENT
PERSISTANT AGENT
SERVER
DBMS
RULES ENGINE
101
103
105
107
109
111
113

FIG._1

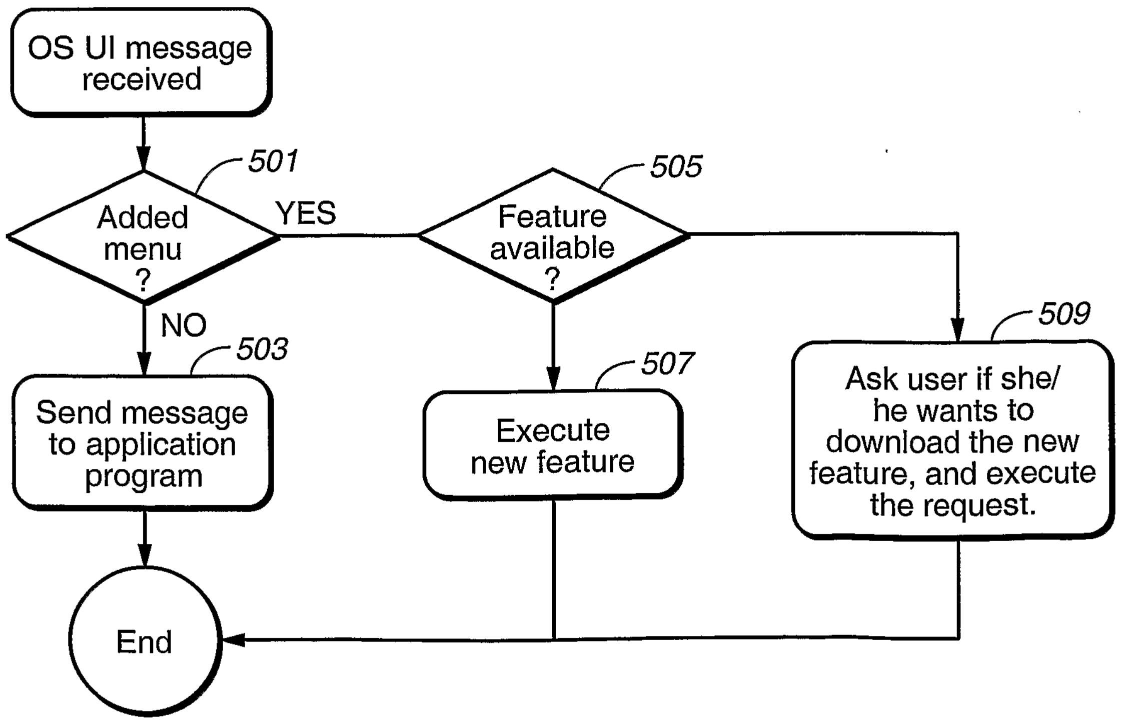
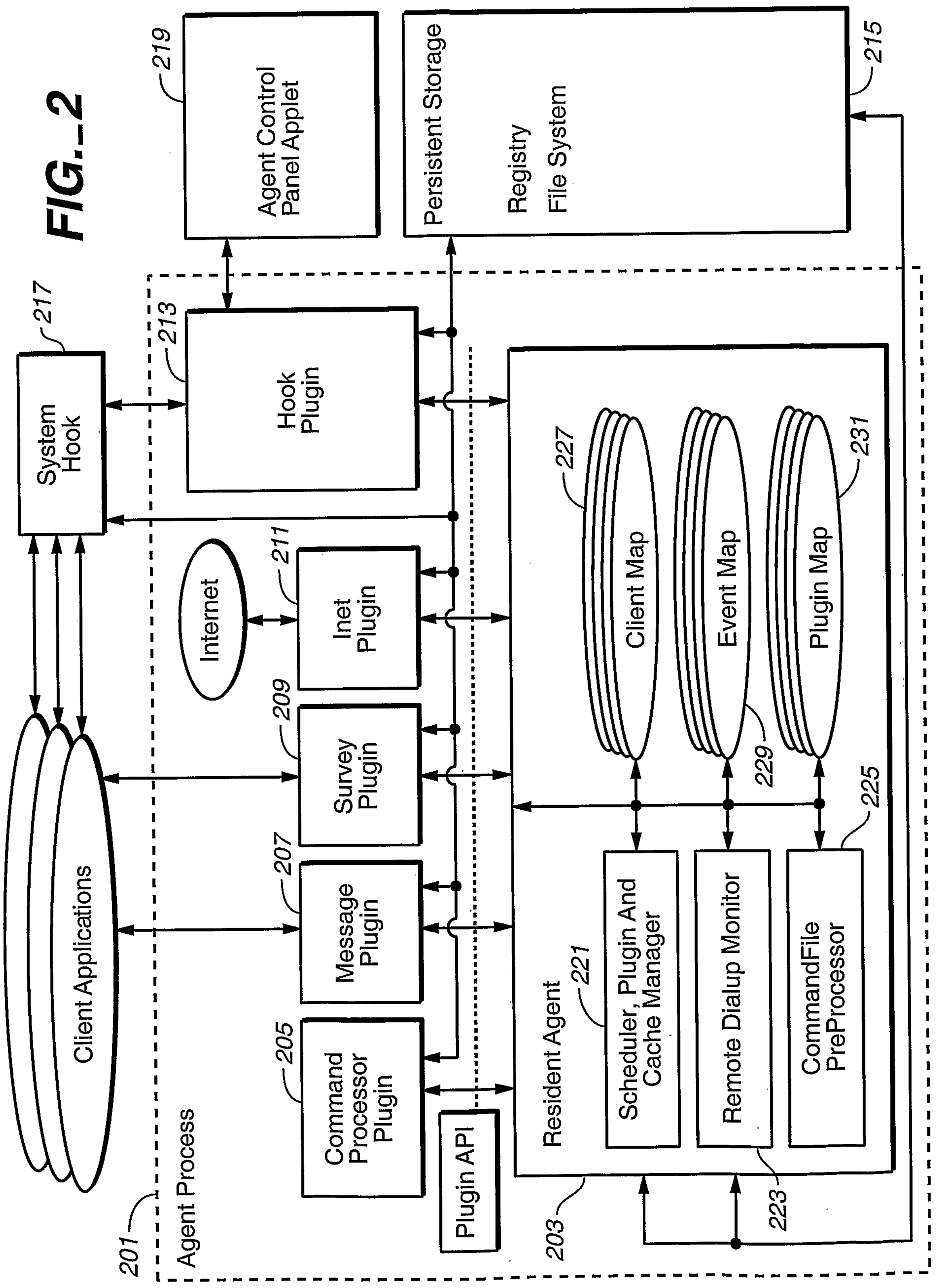

OS UI message received
Added menu ?
501
YES
NO
Send message to application program
503
Feature available ?
505
Execute new feature
507
Ask user if she/ he wants to download the new feature, and execute the request.
509
End

FIG._5

FIG._2


System Hook
217
Agent Control Panel Applet
219
Persistent Storage
Registry
File System
215
Client Applications
Agent Process
201
213
Hook Plugin
Internet
211
Inet Plugin
209
Survey Plugin
207
Message Plugin
205
Command Processor Plugin
Plugin API
Resident Agent
203
227
Client Map
Event Map
229
Plugin Map
231
221
Scheduler, Plugin And Cache Manager
Remote Dialup Monitor
223
CommandFile PreProcessor
225

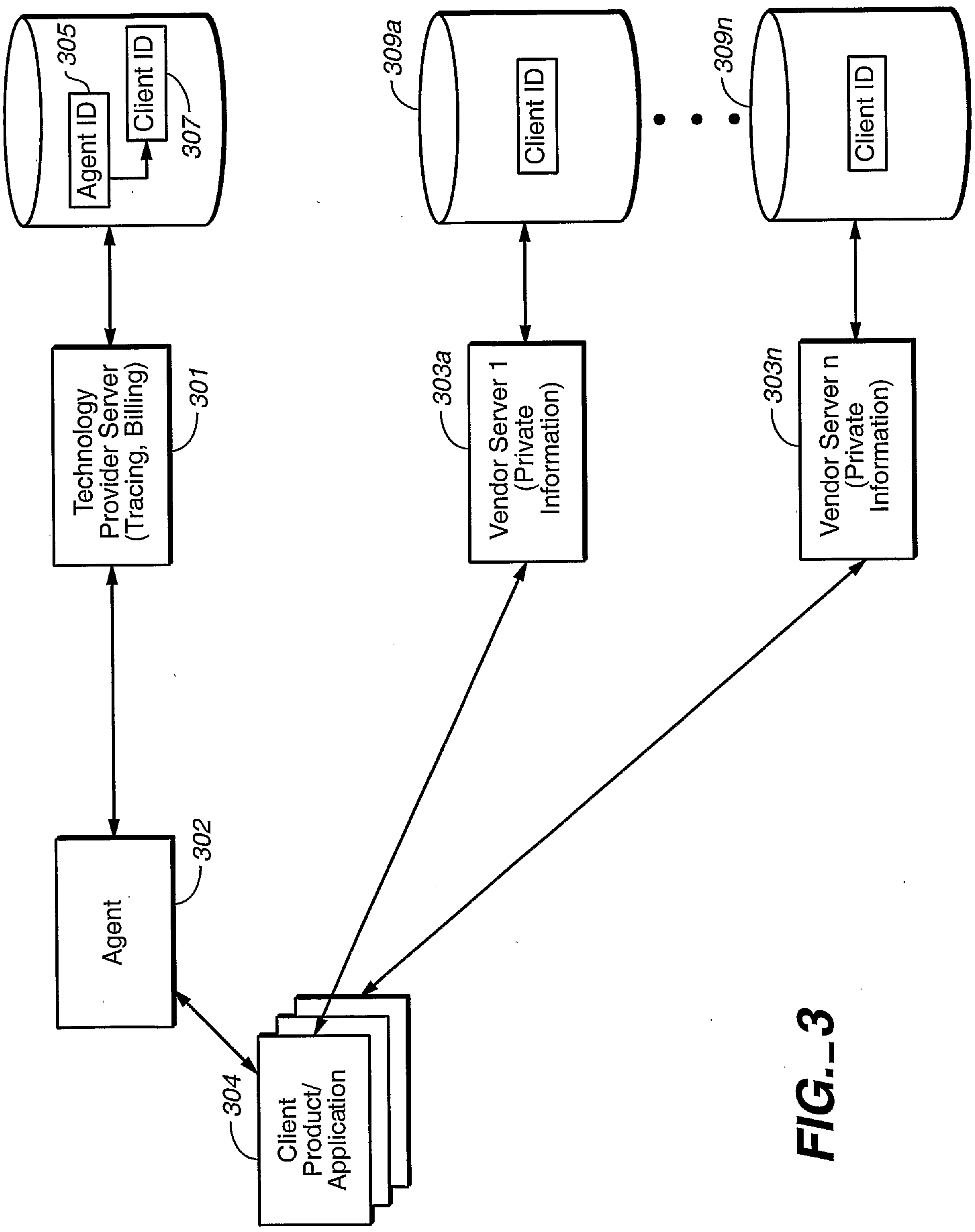
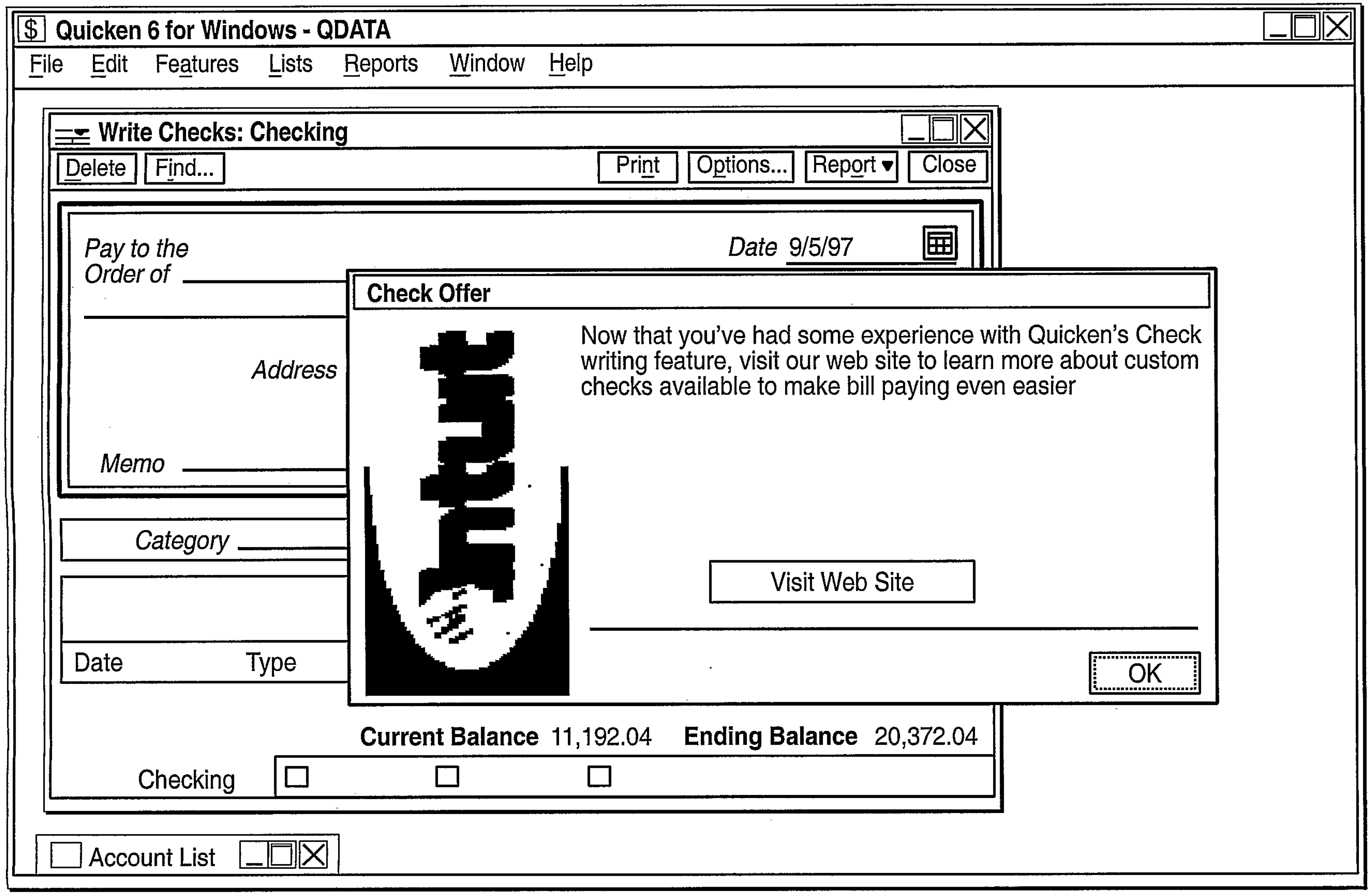

Agent ID
305
Client ID
307
309a
Client ID
309n
Client ID
Technology Provider Server (Tracing, Billing)
301
303a
Vendor Server 1 (Private Information)
303n
Vendor Server n (Private Information)
Agent
302
304
Client Product/ Application

FIG._3


$ Quicken 6 for Windows - QDATA
File Edit Features Lists Reports Window Help
Write Checks: Checking
Delete Find... Print Options... Report ▾ Close
Pay to the Order of
Date 9/5/97
Address
Memo
Category
Date Type
Current Balance 11,192.04 Ending Balance 20,372.04
Checking
Account List
Check Offer
Visit Web Site
OK

Now that you've had some experience with Quicken's Check writing feature, visit our web site to learn more about custom checks available to make bill paying even easier

FIG._4

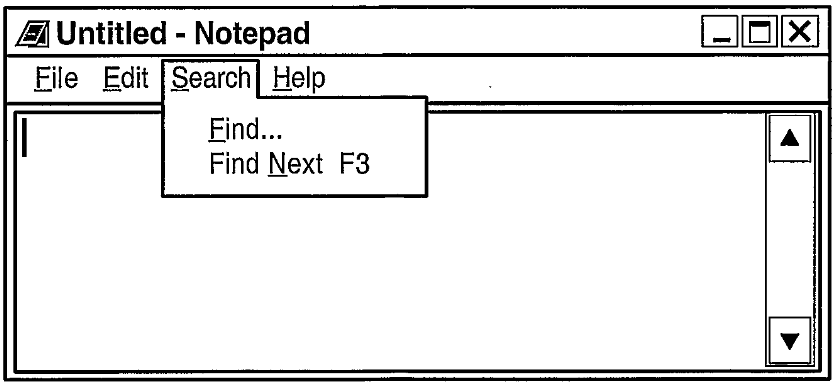


FIG._6

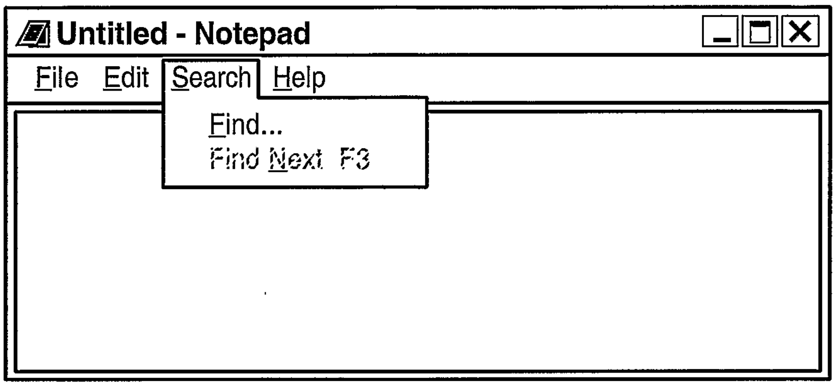
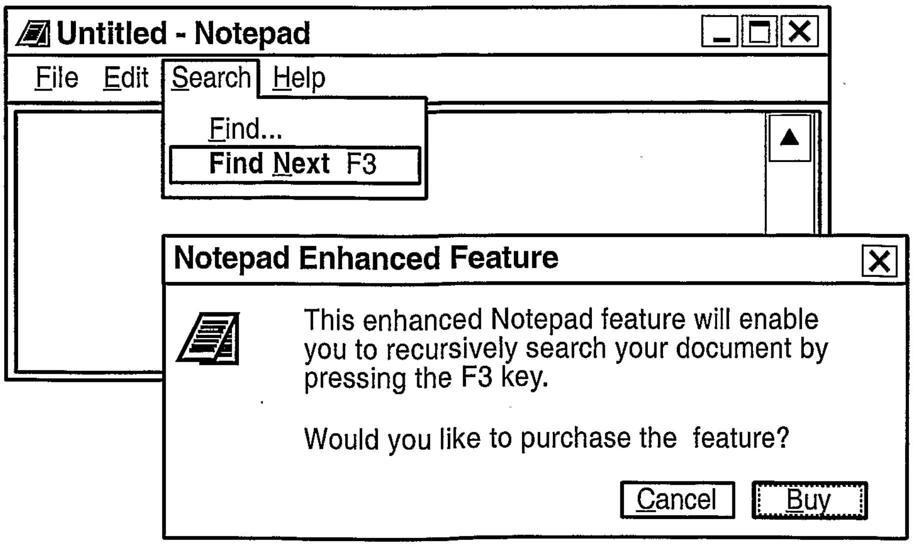


FIG._7



This enhanced Notepad feature will enable you to recursively search your document by pressing the F3 key.

Would you like to purchase the feature?

FIG._8

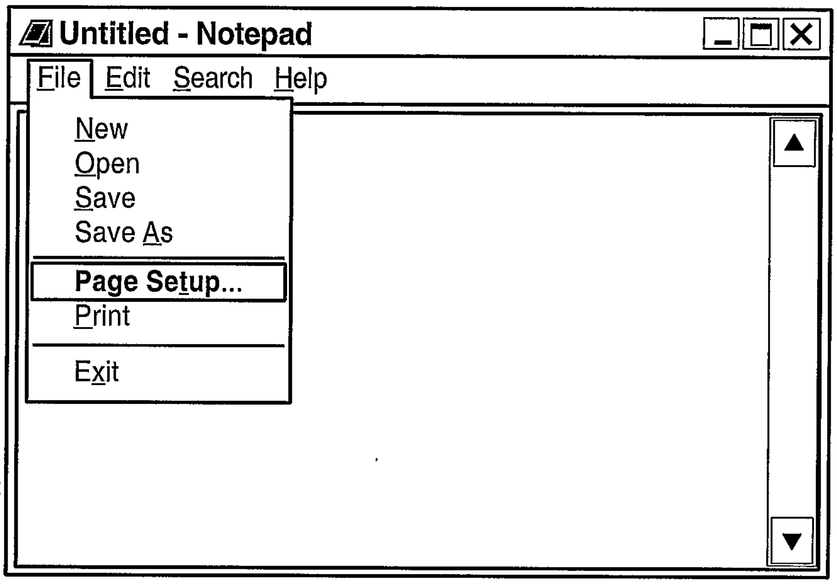

Untitled - Notepad
File Edit Search Help
New
Open
Save
Save As
Page Setup...
Print
Exit

FIG._9

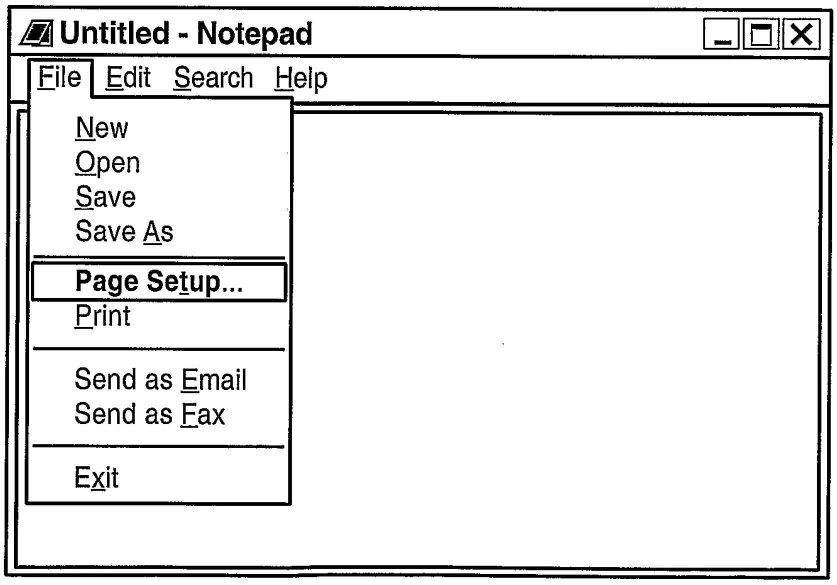

Untitled - Notepad
File Edit Search Help
New
Open
Save
Save As
Page Setup...
Print
Send as Email
Send as Fax
Exit

FIG._10

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 01/42947

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F9/445

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal, IBM-TDB

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	ANONYMOUS: "Privilege User's Guide - Version 2.0" ALADDIN SOFTWARE DOCUMENTATION, 'Online! June 2001 (2001-06), pages c,ix-xii,1-9,11-29,47-75,85-99,101-121,219-233, XP002209452 Retrieved from the Internet: <URL:ftp://ftp.ealaddin.com/pub/privilege/doc/privilegeusersguide.zip> 'retrieved on 2002-08-08! page 2, line 1 -page 3, line 11; figures 1.1,,4.1,15.1 page 4, line 1 -page 6, line 13 page 8, line 1 -page 9, last line page 12, line 1 -page 14, line 7 page 22, line 31 -page 23, line 13 page 47, line 1 -page 49, line 23 page 223, line 18 - line 22 ---	1-6

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search 9 August 2002	Date of mailing of the international search report 22/08/2002
Name and mailing address of the ISA European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Authorized officer Carciofi, A

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	WO 98 58306 A (OYLER SCOTT ;GUTHRIE JOHN (US); TECHWAVE INC (US); KRISHNAN GANAPA) 23 December 1998 (1998-12-23) page 2, line 21 - line 28 page 6, line 11 - line 27 page 7, line 20 -page 8, line 6 page 11, line 16 -page 12, line 24 ---	1-6
A	EP 0 778 512 A (SUN MICROSYSTEMS INC) 11 June 1997 (1997-06-11) column 1, line 19 - line 31 column 9, line 39 -column 10, line 39 column 11, line 47 -column 13, line 8 ---	1-6
A	US 6 243 692 B1 (FLOYD MICHEL ET AL) 5 June 2001 (2001-06-05) column 1, line 50 -column 2, line 41; figure 4 -----	1-6

International Application No
PCT/US 01/42947

Patent document cited in search report		Publication date	Patent family member(s)		Publication date
WO 9858306	A	23-12-1998	US	6073124 A	06-06-2000
			AU	8150598 A	04-01-1999
			WO	9858306 A1	23-12-1998
EP 0778512	A	11-06-1997	US	5708709 A	13-01-1998
			EP	0778512 A2	11-06-1997
			JP	9288575 A	04-11-1997
US 6243692	B1	05-06-2001	WO	9962017 A1	02-12-1999