**PCT** WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/44, 9/46

A1

(11) International Publication Number: **WO 99/63430**

(43) International Publication Date: 9 December 1999 (09.12.99)

(21) International Application Number: PCT/US99/11534

(22) International Filing Date: 25 May 1999 (25.05.99)

(30) Priority Data:
09/086,898 29 May 1998 (29.05.98) US

(71) Applicant: CITRIX SYSTEMS, INC. [US/US]; 6400 N.W. 6th Way, Fort Lauderdale, FL 33309 (US).

(72) Inventors: PANASYUK, Anatoliy; 6/34 Forster Street, West Ryde, NSW 2114 (AU). DUURSMA, Martin; 4 Orchid Place, West Pennant Hills, NSW 2125 (AU).

(74) Agent: RODRIGUEZ, Michael, A.; Testa, Hurwitz & Thibeault, LLP, High Street Tower, 125 High Street, Boston, MA 02110 (US).

(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, 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, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, UG, 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), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

**Published**
*With international search report.*

(54) Title: SYSTEM AND METHOD FOR COMBINING LOCAL AND REMOTE WINDOWS INTO A SINGLE DESKTOP ENVIRONMENT

(57) Abstract

A system for incorporating windows from remote desktop environments into a local desktop environment includes a local node, a local agent, a first remote node, and a first remote agent. The first remote node provides a first remote desktop environment, and the first remote agent monitors the first remote desktop environment for changes in the environment. The first remote node transmits messages to the local agent indicative of changes in the first remote desktop environment. The local agent receives the transmitted messages and commands the local node to modify a representation of a first remote window that is part of a local desktop environment. The local agent also monitors the local desktop and transmits messages to the remote agent indicative of a change in the local desktop. In some embodiment, the local node provides the local desktop environment. Local agents can be embodied on articles of manufacture.

*FOR THE PURPOSES OF INFORMATION ONLY*

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **AL** | Albania | **ES** | Spain | **LS** | Lesotho | **SI** | Slovenia |
| **AM** | Armenia | **FI** | Finland | **LT** | Lithuania | **SK** | Slovakia |
| **AT** | Austria | **FR** | France | **LU** | Luxembourg | **SN** | Senegal |
| **AU** | Australia | **GA** | Gabon | **LV** | Latvia | **SZ** | Swaziland |
| **AZ** | Azerbaijan | **GB** | United Kingdom | **MC** | Monaco | **TD** | Chad |
| **BA** | Bosnia and Herzegovina | **GE** | Georgia | **MD** | Republic of Moldova | **TG** | Togo |
| **BB** | Barbados | **GH** | Ghana | **MG** | Madagascar | **TJ** | Tajikistan |
| **BE** | Belgium | **GN** | Guinea | **MK** | The former Yugoslav Republic of Macedonia | **TM** | Turkmenistan |
| **BF** | Burkina Faso | **GR** | Greece | | | **TR** | Turkey |
| **BG** | Bulgaria | **HU** | Hungary | **ML** | Mali | **TT** | Trinidad and Tobago |
| **BJ** | Benin | **IE** | Ireland | **MN** | Mongolia | **UA** | Ukraine |
| **BR** | Brazil | **IL** | Israel | **MR** | Mauritania | **UG** | Uganda |
| **BY** | Belarus | **IS** | Iceland | **MW** | Malawi | **US** | United States of America |
| **CA** | Canada | **IT** | Italy | **MX** | Mexico | **UZ** | Uzbekistan |
| **CF** | Central African Republic | **JP** | Japan | **NE** | Niger | **VN** | Viet Nam |
| **CG** | Congo | **KE** | Kenya | **NL** | Netherlands | **YU** | Yugoslavia |
| **CH** | Switzerland | **KG** | Kyrgyzstan | **NO** | Norway | **ZW** | Zimbabwe |
| **CI** | Côte d'Ivoire | **KP** | Democratic People's Republic of Korea | **NZ** | New Zealand | | |
| **CM** | Cameroon | | | **PL** | Poland | | |
| **CN** | China | **KR** | Republic of Korea | **PT** | Portugal | | |
| **CU** | Cuba | **KZ** | Kazakstan | **RO** | Romania | | |
| **CZ** | Czech Republic | **LC** | Saint Lucia | **RU** | Russian Federation | | |
| **DE** | Germany | **LI** | Liechtenstein | **SD** | Sudan | | |
| **DK** | Denmark | **LK** | Sri Lanka | **SE** | Sweden | | |
| **EE** | Estonia | **LR** | Liberia | **SG** | Singapore | | |

# SYSTEM AND METHOD FOR COMBINING LOCAL AND REMOTE WINDOWS INTO A SINGLE DESKTOP ENVIRONMENT

## Field of the Invention

The present invention relates to displaying information on remote computers and, in particular, to a system and method for combining display data received from various remote sources into a single, local display.

## Background of the Invention

Client-server systems, in which a user of a client node is typically remote from a server which provides application processing or access to files and other resources, are both convenient and cost-effective. Client nodes are generally cheaper than servers, and since one server typically provides services to more than one client, overall system cost is reduced. Additionally, client-server systems allow an enterprise to make decisions regarding the location of certain system resources (such as applications) on a situational basis. For example, certain applications may be resident solely on clients, solely on servers, solely on certain servers, or any combination of the above which improves the overall efficiency of the system.

To date, however, efforts to combine output data from various sources into a single display have not met with success. For example, early attempts have been made to cause server-based applications to write directly into local windows. Although this method can display application output from various servers on a single display, it lacks the ability to arrange the windows on the client responsive to the z-axis ordering of the windows at each individual server. Thus, if a server brings a new window to the top of its desktop, no corresponding change appears to the user at the client.

Further, systems typically cannot support combining various sources of data into a single display without modification of the applications generating output data. This results because most enterprises desire to use off-the-shelf software to generate output data and such software does not support combination of output data. This represents a practical problem because re-writing such applications to support output combination is generally prohibited by the manufacturer of such software and, even if not prohibited, can be expensive.

Summary of the Invention

The present invention relates to a system in which multiple data displays can be represented as a cohesive, single, unitary display, without intervention on the part of the user and without requiring modification of the applications generating displayed output data. The system allows a user to interact with displayed windows without knowledge of the source of those windows, and changes to the window, either locally or remotely, are reflected in the corresponding display on the server or client.

In one aspect, the present invention relates to a system for incorporating windows from remote desktop environments into a local desktop environment. The system includes a local node, a local agent, a first remote node, and a first remote agent. The first remote node provides a first remote desktop environment, and the first remote agent monitors the first remote desktop environment for changes in the environment. The first remote node transmits messages to the local agent indicative of changes in the first remote desktop environment. The local agent receives the transmitted messages and commands the local node to modify a representation of a first remote window that is part of a local desktop environment. The local agent also monitors the local desktop and transmits messages to the remote agent indicative of a change in the local desktop. In some embodiments, the local node provides the local desktop environment.

In another aspect, the present invention relates to a method for incorporating windows from remote desktop environments into a local desktop environment. The method comprises the steps of: providing a local node hosting a local agent; receiving, by the local agent, a message indicating a change to windows included in a remote desktop environment; commanding, by the local agent, the local node to effect a corresponding change in the local desktop environment; monitoring, by the local agent, the local desktop; and transmitting, by the local node, messages to the remote node indicative of a change in the local desktop environment. The method may be embodied on an article of manufacture.

In yet another aspect, the present invention relates to an agent which incorporates windows from remote desktop environments into a local desktop environment. The agent includes a message receiving process capable of receiving messages indicating a change has occurred in a remote desktop environment. A command process effects changes to the local desktop environment responsive to messages received by the message receiving process. A monitor process monitors local desktop events. A transmission process transmits messages

indicating occurrence of the local desktop event. The agent may be embodied on an article of manufacture.

In a further aspect, the present invention relates to a system for incorporating windows from a remote desktop into a local desktop. The system comprises a local node and a remote node connected by a communications link. The communications link includes a first virtual channel and a second virtual channel. The nodes exchange desktop information such as window position, window size, and z-ordering of desktop windows, over the first virtual channel. The nodes exchange graphical information over the second virtual channel. In some embodiments, the first virtual channel and the second channel may be provided as a single virtual channel.

Brief Description of the Drawings

The invention is pointed out with particularity in the appended claims. The advantages of this invention described above, and further advantages, may be better understood by reference to the following description taken in conjunction with the accompanying drawings, in which:

FIG. 1 is a functional block diagram of an embodiment of a client-server system;

FIG. 2 is a functional block diagram of a client node connected to two separate server nodes;

FIG. 3 is a flow diagram of the steps to be taken when a server agent detects a change in its associated desktop environment;

FIG. 4 is a flow diagram of the steps to be taken when a client agent detects a change in its associated desktop environment;

FIG. 5 is a flow chart of the steps to be taken to open a virtual channel between a client agent and a server agent; and

FIG. 6 is a functional block diagram of an agent.

Detailed Description of the Invention

Referring now to Fig. 1, a client-server system is shown in which server node 20 provides services for one or more client nodes 10. Client nodes may communicate with server node 20 via any of a number of industry-standard data communications protocols including, but not limited to, TCP/IP, IPX/SPX, NetBEUI, or serial protocols. Alternatively, client nodes 10 may connect to server node 20 using a proprietary data communications protocol such as the ICA protocol manufactured by Citrix Systems, Inc. of Fort Lauderdale, Florida or the RDP protocol

manufactured by Microsoft Corporation of Redmond, Washington. The actual connection between the client nodes 10 and the server node 20 may be physical cabling or it may be a wireless connection, such as infrared transmission.

Fig. 2 depicts a system in which a single client node 10 is connected to more than one server node 20, 20'. As shown in Fig. 2, client node 10 has an associated display 12. The display 12 may be used to display one or more components of a graphical user interface, such as windows and pull-down menus. The collection of graphical user interface components displayed to a user by the display 12 is generally referred to as the "desktop." As shown in Fig. 2, the local node 10 displays a local desktop environment 14 to a user. Local node 10 may provide at least a part of the local desktop environment 14 or local node 10 may simply display various desktop components received from other sources such as server nodes 20. As shown in Fig. 2, each server node 20, 20' has an associated display 22, 22' which also displays a desktop environment 24, 24'. It should be noted that display 22, 22' need not be a video display monitor. For example, display 22, 22' may simply be a bank of video RAM to which applications write the output of graphical procedure calls. Fig. 2 depicts an embodiment of a system in which each server node display 22, 22' displays one graphical user interface window 26, 27'.

Each server 20, 20' also includes at least one agent 30, 30'. In particular embodiments, each server 20, 20' includes one agent 30, 30' for each client 10 connected to the server 20, 20'. Client node 10 may also host an agent 40. In some embodiments, a client node 10 hosts a separate local agent 40 for each server to which the client node 10 is connected. In other embodiments, the client node 10 hosts a single agent 40 that manages connections to multiple server nodes 20. Each of the agents 30, 30', 40 may monitor their associated desktop environment 24, 24', 14 for windows which: change position; are opened; are closed; change size; are minimized; are maximized; or are brought to the top of the desktop, i.e., windows which gain focus that do not previously have focus. Each agent 30, 30', 40 transmits messages indicative of changes in their associated desktop 24, 24', 14 to other agents. For example, local agent 40 may receive messages transmitted from server node agents 30, 30'. The local agent 40 commands the client 10 to modify the local desktop environment 14 in response to the messages received from server agents 30, 30', that is, the local agent 40 issues commands to the client node 10 to conform the local desktop environment 14 to the server desktop environment 24. In other embodiments, server node agents 30, 30' receive messages from a local agent 40 and

command the server 20, 20' to modify the server desktop environment 24, 24' in response to messages received from the local agent 40.

In one embodiment, the agents 30, 40 monitor changes to their associated desktop environment 24, 24' by periodically issuing one or more of a set of commands provided by the operating system that allow details of the graphical user interface desktop to be determined. For embodiments in which the agents 30, 40 reside on nodes that execute a version of the WINDOWS operating system, the agents 30, 40 may periodically issue the EnumWindows command to the WINDOWS operating system, which returns a list of all windows present on the desktop, together with information related to those windows. The agents 30, 40 can issue the EnumWindows command every 50 milliseconds, every 100 milliseconds, every 500 milliseconds, or at any period that allows the agent 30, 40 to rapidly determine when changes to its associated desktop environment have occurred without putting a significant computational burden on the node. In this embodiment, the agent 30, 40 maintains a data structure storing information about the desktop windows and compares the values returned by the EnumWindows command to the data structure to determine changes.

Information determined and stored by the agent 30, 30' can include the title bar associated with each window, the location of each window in the desktop environment 24, 24', the size of each window, and the z-order positioning of each window in the desktop environment 24, 24'. In another embodiment, the agent 30, 30', 40 monitors an intranode graphics message queue to determine changes to its associated desktop environment. Server agents 30, 30' monitor an intraserver message queue and local agent 40 monitors an intraclient message queue. In this embodiment, changes to the desktop environment 24, 24' are effected via messages sent to a graphics subsystem from system applications or the operating system itself. Thus, an application executing an a server 20, 20' would send a message to a graphics engine residing on the server 20, 20' in order to change the server desktop environment 24, 24'. Other commands which return graphical user interface data are readily apparent to those of ordinary skill in the art. For embodiments in which the agents 30, 40 reside on nodes executing a version of the WINDOWS operating system, the agents 30, 40 monitor the Windows Message Queue for messages affecting the desktop environment associated with the node on which the agent resides. Examples of such messages include: WM_SETFOCUS, which indicates to which window focus will be given (i.e., brought to the "top" of the desktop); WM_KILLFOCUS, which removes

focus from an indicated window; and WM_WINDOWPOSCHANGING, which indicates a change in the position of a window. Other messages that can be posted to the Windows Message Queue are readily known to those of ordinary skill in the art.

Referring now to Fig. 3, the steps taken during a server-initiated event are shown. The server agent 30 senses a change in its associated desktop (step 302). The server agent 30 may do this by intercepting a window event on the server message queue, or the agent 30 may determine a change in the desktop by comparing the results returned from serially issued operating system commands, as described above. The server agent 30 sends a message to a client agent 40 indicating the change in the server desktop 24 (step 304). For example, if a new window has been given focus, the server agent 30 can transmit a message to a client agent 40 indicating the identity of the new "top" window. In one embodiment, the server agent 30 broadcasts its message to all client agents 40 that exist in the system. Alternatively, the server agent 30 may transmit its message only to a predetermined subset of client agents 40. For example, when a client 10 makes a connection to a server 20, the client agent 40 may register with the server agent 30. In this embodiment, the server agent 30 would transmit change messages only to those client agents that have registered with the server.

The client agent 40 receives the transmitted message (step 306). In embodiments in which the server broadcasts commands, the client agent 40 must have some mechanism for determining whether a transmitted command affects its associated desktop 12. For example, the client agent 40 may maintain a list of servers to which it is connected. In these embodiments, the client agent 40 responds to messages broadcast by any server present in its list. For embodiments in which the server agent 30 does not broadcast messages, no such mechanism is necessary.

The client agent 40 implements a change to its associated desktop 14 responsively to the received message (step 308). The client agent 40 may accomplish this by directly issuing graphics Application Programming Interface commands that cause the client 10 to change the display of its associated desktop 14. Alternatively, the client agent 40 may issue GDI commands to change its associated desktop 14. In still other embodiments, the client agent 40 issues commands directly to the system, whether implemented in hardware or software, responsible for displaying graphics on the client 10.

Referring now to Fig. 4, the steps taken when a client initiates a desktop change are shown. The client agent 40 senses a change in its associated desktop 14 (step 402). As noted above, this may be done on an event-driven basis or by polling the operating system operating on the client 10. The client agent 40 determines to which server 20 the affected window belongs (step 404). To facilitate this process, the client agent 40 may maintain a list that associates remote windows with a particular server 20. The client agent 40 then sends a message to the identified server 40 indicating the change in its desktop 14 (step 406). Alternatively, the client agent 40 may skip step 404 entirely and broadcast its change message to all servers 20. The server agent receives the transmitted message (step 408) and implements the change in its associated desktop (step 410), as described above.

In one particular embodiment, a client node 10 and a server node 20 communicate using the ICA protocol and the client node 10 and the server node 20 execute a version of the WINDOWS operating system. Client node 10 hosts a local agent 40 that may be provided as a dynamically linked library module. The server node 20 hosts a server agent 30 that may be provided as a separate thread.

In this embodiment, the local agent 40 and the server agent 30 exchange graphical data, i.e., the data actually displayed in each window on the desktop, via a first ICA virtual channel. Information about window positioning, window size, z-access ordering of window and other such information is communicated between the client node 10 and the server node 20 via a second ICA virtual channel. Throughout the description, when the client node 10 and the server node 20 are actively exchanging information via the second ICA virtual channel, the client will be referred to as being in "seamless windowing mode."

Referring now to Fig. 5, the process for enabling seamless windowing mode between the local agent 40 and server agent 30 is shown. In this embodiment, all communication between a server agent and a client agent is packet-oriented and takes place over a dedicated ICA virtual channel, making the functioning of the agents 30, 40 independent from the underlying communication protocol. All packets start with packet type (1 byte), followed by packet data length (2 bytes, can be zero) and data (optional). Agents 30, 40 will try to send as much data in a single network packet as possible, but it will always send complete packets. That is, the size of seamless window virtual packets never exceeds the allowable size of an ICA packet. Packet flow

control and delivery confirmation is implemented by the transport level of the ICA protocol. Individual packets are executed immediately on reception.

The client agent 40 waits for an initial packet from the server agent 30. After user logon to the server, a server agent 30 will be invoked (step 504).

The server agent sends a TWI_PACKET_START packet to the client agent 40, which includes some essential information about the server desktop environment (desktop resolution, desktop size, version number of ICA protocol supported by the server, etc.) (step 506). This packet is sent by the server agent 30 on initial connection or on reconnect, and is used to: (1) detect seamless windowing capabilities of the client; and (2) requests basic client information.

The client agent receives the TWI_PACKET_START packet (step 507) and responds with a TWI_PACKET_C2H_START_ACK packet, confirming TWI_PACKET_START and supplying client version/capabilities information (step 508). This packet is sent by the client agent 40 to confirm reception of TWI_PACKET_START packet and to send the requested basic client information to the server agent 30.

If there is no response from the client agent 40 (step 509), the server agent 30 assumes that the client is unable to enter seamless windowing mode, and the seamless windowing virtual channel is not used by the server node 20 to communicate window information. In this case, the server node 20 continues to communicate graphical data to the client node 10 via another virtual channel, and the client desktop displays the server desktop without incorporating windows from other nodes.

The client agent 40 uses the information sent by the server agent 30 in step 506 to determine if a seamless windowing session can be established between the server agent 30 and the client agent 40. In one embodiment, the client agent 40 compares information relating to the version of the virtual channel protocol supported by the server agent 30 to makes the determination If the client agent 40 determines that it is possible to enable seamless windowing mode (step 510), the client agent 40 sends a TWI_PACKET_C2H_OPEN packet to the server agent 30 (step 511). This packet requests that the server agent 30 enable seamless windowing mode.

On reception of a TWI_PACKET_C2H_OPEN packet (step 512) the server agent 40 (I) resets its internal data structures, (ii) sends a TWI_PACKET_SYSINFO packet to the client agent 40 to communicate some general information regarding the window settings on the server

node 20 to the client agent 40, (iii) sends a TWI_PACKET_OPEN packet to the client agent 40 (step 514) indicating the establishment of seamless windowing mode, and (iv) enables its main polling loop (step 516) that will poll the operating system on the server node for desktop changes. If the client agent 40 and the server agent 30 do not support the same version of the seamless window protocol, the server agent 30 ignores the TWI_PACKET_C2H_OPEN packet.

On reception of TWI_PACKET_OPEN packet (step 520), the client agent 40 resets its internal data structures (step 522) and seamless windowing mode between the client agent 40 and the server agent 30 is established.

During a seamless windowing mode session, the server agent 30 will send window information such as window position, size, styles, window text, etc. for all top-level windows on the server node. Also, foreground window information is sent, i.e., which window on the server node desktop is the foreground window. In accordance with this information, the client agent 40 creates windows with the same size/position as the server node windows on the client node desktop. In some embodiments, window elements are transmitted as bitmaps from the server node 20. Examples of packets sent by the server agent 30 include : TWI_PACKET_CLOSE, which is sent to switch the client agent 40 out of seamless windowing mode and back to regular, or full screen, mode; that is, the client node 10 is switched back to displaying the server node desktop environment without incorporating windows from other desktop environments; TWI_PACKET_CREATEW, which is sent to create new windows on the client node 10; TWI_PACKET_DELETEW, which is sent to destroy a window on the client node 10; TWI_PACKET_CHANGEW, which is sent to change a window displayed by the local node 10; TWI_PACKET_SYSINFO, which is sent to report server node 20 system settings -- normally it is sent only once, but the packet can be sent multiple times; TWI_PACKET_FOREGROUNDW, which is sent during normal seamless windowing mode operation to change the foreground window; TWI_PACKET_SETTOPW, which is sent during normal seamless windowing mode operation to change the top window, that is, to bring a new window to top; TWI_PACKET_SETFOCUS, which is sent during normal seamless windowing mode operation to change the focus window; TWI_PACKET_FOCUSACK, which is sent in response to TWI_PACKET_C2H_SETFOCUS (see below), and reports the result of a SetFocus attempt; and TWI_PACKET_SPA_STATUS, which is sent in response to

TWI_PACKET_C2H_START_PUBLICAPP (see below), and is used to report the result of the requested operation.

Examples of packets that can be sent by the client agent 40 to the server agent 30 include : TWI_PACKET_C2H_PAUSE, which is sent to suspend the server agent 30, that is, the server agent 30 will stop sending window information, clear its internal data structure and send a TWI_PACKET_CLOSE packet (see above); TWI_PACKET_C2H_RESUME, which is sent to resume the server agent 30 -- the server agent 30 will clear its internal data structure, and send a TWI_PACKET_OPEN packet (see above); TWI_PACKET_C2H_SETPOS, which is sent to report window size/position change on the client node; TWI_PACKET_C2H_SETFOCUS, which is sent to report a change in the focus window on the client node; TWI_PACKET_C2H_RESTORE, which is sent to request restoration of a minimized window; TWI_PACKET_C2H_TERMINATE, which is sent to request termination of a program executing on the server node 20; TWI_PACKET_C2H_STARTAPP, which is sent to start a new application on the server node 20; TWI_PACKET_C2H_LOGOUT, which is sent to end the current session; TWI_PACKET_C2H_START_PUBLICAPP, which is sent to start a new published application on the server node; and TWI_PACKET_C2H_CLIENTINFO, which is sent to report client desktop settings to the server agent 30 -- this packet is generally sent on startup, but can also be used during seamless windowing session.

The client agent 40 will try to perform some operations (such as window move and resize) locally, sending update information back to the server node 40 afterwards. Proper window behavior is emulated by intercepting the WM_NCHITTEST message for the client-created windows.

Foreground window changes can happen on both the client node and the server node, so the client and server will negotiate and balance actual foreground window changes. For example, if the server node 20 changes its foreground window, that change should be properly represented on the client desktop. The server agent 30 sends information regarding the new foreground window to the client agent 40 using the TWI_PACKET_FOREGROUNDW packet. Similarly, if the client agent 40 detects a foreground window change on the client desktop, the client agent 40 sends information regarding the change to the server agent 30 and the server agent 30 implements the change on the server desktop.

When focus is taken away from a window representing a server window and is given to a local client window, the client notifies the server of the change and the server gives focus to an invisible window. For embodiments in which the client node 10 is connected to two server nodes 20, and focus is shifted from a window representing a window from the first server and is given to a window representing a window from the second server, the client sends a packet informing the current server that its window no longer has focus. Once the server responds by giving focus to an invisible window, the client agent 40 instructs the other server that its window now has focus on the local desktop.

In some embodiments, it is desirable to add some complexity to the agent's main polling loop to reduce network traffic. In these embodiments, the main polling loop includes a comparison between the current foreground window and the identity of the window last requested to be moved to the foreground. If the current foreground window matches the window identified in the most recent request, the agent does not need to send information acknowledging the change. This technique is useful in both server agent 30 and client agents 40.

Window z-ordering on the client is a superset of the server node z-ordering (client will always have more windows than the host). Server node z-ordering is reproduced on the client by reproducing owner/owned relationship among windows and the TOP_MOST flag in the window style. Owner/owned relationships refer to windows which are children of other windows, such as dialog boxes associated with application windows. The dialog box is said to be owned by the application window, and the dialog box will always appear on top of its owner. The TOP_MOST flag indicates that a particular window should appear on "top" of the desktop, for example, the status bar in WINDOWS 95.

When a user disconnects, the server agent 30 switches itself to suspended mode, and will not send information to the client agent 40. On a reconnect, the server agent 30 sends a TWI_PACKET_START packet, reporting HostAgentState as "already running, reconnect."

Based on the version number of the protocol supported by the server the client will decide whether it is possible to enable seamless windowing mode (from the client point of view). If it is possible to switch to seamless windowing mode, the client agent 40 will send a TWI_PACKET_C2H_OPEN packet, asking the server agent 30 to enable seamless windowing mode.

Each agent responsible for monitoring an associated desktop may be implemented as a stand-alone software routine (such as an executable file on DOS-based systems), a dynamically linked library routine (DLL), or as an integral piece of the operating system. Referring now to FIG. 6, and in brief overview, each agent includes a message receiving facility 602, a command facility 604, a monitor facility 606, and a message transmission facility 608. Agent-agent communication is full-duplex, i.e., agents can transmit and receive messages simultaneously. Thus, each facility can be implemented as a separately functioning code segment that operates independently of the other facilities. For example, message receiving facility 602 and command facility 604 can be implemented as separate threads which communicate with each other via a named pipe or shared memory. Use of a common data allows the message receiving facility 602 and the message transmitting facility 608 to be synchronized.

Message receiving facility 602 receives messages transmitted from other agents indicating changes in the desktop environments associated with those agents. Message receiving facility 602 may connect directly with the physical layer of the communications protocol the agents use to communicate, or the message receiving facility 602 may operate at a higher layer of the protocol by cooperating with one or more communications subsystems. For embodiments in which messages are broadcast by agents, the message receiving facility 602 has some mechanism for determining whether a broadcast message is intended for it. For example, the message receiving facility 602 may store a list of the windows which its associated desktop displays. The message receiving facility 602 would compare the target of any received message to its list of windows to determine whether or not to take action on the received message. The message receiving facility may be implemented as a blocking function. Alternatively, the message receiving facility can be implemented a call-back function invoked by the ICA virtual channel transport.

Once the message receiving facility 602 has determined that a received message is intended for its desktop, the command facility is invoked to effect the change indicated by the message to the associated desktop environment. The command facility 604 may be passed the received message facility, or the message receiving facility 602 may process the received message before communicating with the command facility 604. The command facility 604 may implement the desktop change indicated by the received message by issuing GDI commands. In

other embodiments, the command facility 604 may issue commands directly to an associated graphics subsystem or may issue other graphics API commands.

During a seamless windowing session, a number of desktops are associated with a single client node - one desktop on the client itself and one desktop per server node 20 to which the client node 10 is connected. The client agent 40, in conjunction with the server agent 30, 30', creates a combined window list representing the z-order of all desktops. All participating desktops are "linked" together by the client agents 40 and the server agents 30, 30', and any z-order changes on any desktops will be propagated to other desktops.

In one embodiment, each server has knowledge only of its own graphical desktop representation and the server desktops are individually represented within the client. The client display is updated by combining all server and client desktop images into a single display image based on the window information that has been obtained from each server node 20, 20' by the client agent 40. The resulting image is displayed at the client node 10.

The combining process involves building a common window list based on the windows information exchanged by all agents. Using the combined window list, the graphical desktop data is clipped and merged for representation by the client node 10. The node takes care of "clipping" displayed windows resulting from the commands issued by the command facility 604. Such "clipping" functions are well-known to those of ordinary skill in the art. In some embodiments, however, the command facility 604 maintains a shadow bitmap of clipped windows. That is, the command facility 604 maintains a bit image of windows that are obscured by other windows. This allows the agent to change its associated desktop without requiring it to reload the window image of an obscured window from the appropriate source. In other embodiments, the node determines whether graphical data is obscured at the time it is received. If it is, the node ignores the received graphical data. If it is not, the node displays the data. The node makes a determination as to whether the graphical data is obscured by applying clipping functions.

Monitoring facility 606 monitors the desktop associated with the agent. Monitoring facility 606 may monitor the desktop by periodically issuing commands provided by the operating system executing on the node which return information about the node's desktop. Alternatively, the monitoring facility 506 may watch for messages posted to an intranode message queue. As noted above, in one particular embodiment the monitoring facility 606

monitors the Windows Message Queue. Once a desktop change occurred, the message transmission facility 608 transmits a message indicating the change that has occurred. In some embodiments, the message transmission facility 608 broadcasts notification of the change.

In one embodiment, message transmission facility 608 can be implemented in the form of non-blocking function, that can be called from any window procedure. If the function can not send a data packet immediately (for example, the communication subsystem has no buffer space), a timer will be set and retry attempts will be done until the send succeeds.

The present invention may be provided as one or more computer-readable programs embodied on or in one or more articles of manufacture. The article of manufacture may be a floppy disk, a hard disk, a CD ROM, a flash memory card, a PROM, a RAM, a ROM, or a magnetic tape. In general, the computer-readable programs may be implemented in any programming language. Some examples of languages that can be used include C, C++, or JAVA. The software programs may be stored on or in one or more articles of manufacture as object code.

Having described certain embodiments of the invention, it will now become apparent to one of skill in the art that other embodiments incorporating the concepts of the invention may be used. Therefore, the invention should not be limited to certain embodiments, but rather should be limited only by the spirit and scope of the following claims.

What is claimed is:

1. A system for incorporating windows from remote desktop environments into a local desktop environment, the system comprising:

a local node having a local desktop environment;

a local agent;

a first remote node having a first remote desktop environment including at least one first remote window; and

a first remote agent monitoring said first remote desktop environment and in communication with said local agent, said first remote agent transmitting a message to said local agent indicative of a change to said first remote window,

said local agent receiving the message transmitted by said first remote agent and, responsive to the received message, commanding said local node to modify a representation of said first remote window as part of said local desktop environment and said local agent transmitting messages to said first remote agent indicative of a locally-generated change to a representation of said first remote window.

2. The system of claim 1 wherein said local node provides said local desktop environment, said local agent monitors said local desktop environment, and wherein said local agent receives the messages transmitted by said remote agent and, responsive to the received message, commands said local node to modify a representation of said first remote window as part of said local desktop environment.

3. The system of claim 1 wherein said first remote agent transmits a message indicating that said first remote window has changed position and wherein said local agent receives the message transmitted by said first remote agent and commands said local node to change the position of a representation of said first remote window in said local desktop environment.

4. The system of claim 1 wherein said first remote agent transmits a message indicating that said first remote window has closed and wherein said local agent receives the message transmitted by said first remote agent and commands said local node to close a representation of said first remote window in said local desktop environment.

5. The system of claim 1 wherein said first remote agent transmits a message indicating that said first remote window has changed size and wherein said local agent receives the transmitted message and commands said local node to change the size of a representation of said first remote window in said local desktop environment.

6. The system of claim 1 wherein said first remote agent monitors said first remote desktop environment by monitoring an intranode message queue.

7. The system of claim 1 wherein said local node provides said local desktop environment and said local agent monitors said local desktop environment.

8. The system of claim 7 wherein said local node broadcasts a message indicative of a locally-generated change to a representation of said first remote window.

9. The system of claim 1 wherein said remote node broadcasts a message indicative of a change to said first remote window.

10. The system of claim 1 wherein said local node performs clipping functions on said representations of said first remote window that are obscured by local windows or other remote windows.

11. A method for incorporating windows from remote desktop environments into a local desktop environment, the method comprising the steps of:

(a) providing a local node hosting a local agent;

(b) receiving, by the local agent, a message indicating a change to windows included in a remote desktop environment;

(c) commanding, by the local agent, the local node to effect a corresponding change in the local desktop environment;

(d) monitoring, by the local agent, the local desktop environment; and

(e) transmitting, by the local agent, a message indicative of a change to representation of the remote windows included in the local desktop environment.

12. The method of claim 11 wherein steps (b) and (c) comprise:

(b) receiving, by the local agent, a message indicating that a window included in a

remote desktop environment has changed position; and

(c) commanding, by the local agent, the local node to change the position of a representation of the remote window in the local desktop environment.

13. The method of claim 11 wherein steps (b) and (c) comprise:

(b) receiving, by the local agent, a message indicating that a window included in a remote desktop environment has closed; and

(c) commanding, by the local agent, the local node to close a representation of the remote window in the local desktop environment.

14. The method of claim 11 wherein steps (b) and (c) comprise:

(b) receiving, by the local agent, a message indicating that a window included in a remote desktop environment has changed size; and

(c) commanding, by the local agent, the local node to change the size of a representation of the remote window in the local desktop environment.

15. An agent for incorporating windows from remote desktop environments into a local desktop environment, the agent comprising:

a message receiving process capable of receiving messages indicative of a change in a remote desktop environment;

a command process capable of effecting changes to a local desktop environment responsive to the received messages;

a monitor process capable of monitoring local desktop events; and

a transmission process capable of transmitting messages indicative of the local desktop events.

16. The agent of claim 15 wherein said transmission process broadcasts messages indicative of the local desktop events.

17. The agent of claim 15 wherein said command process issues application programming interface commands, responsively to received messages, to effect changes to a local desktop environment.

18. A article of manufacture having the agent of claim 15 embodied thereon.

19. A system for incorporating windows from remote desktop environments into a local desktop environment, each desktop environment including at least one window displaying graphical data, the system comprising:

a communications link comprising a first virtual channel and a second virtual channel;

a local node; and

a remote node,

said local node and said remote node exchanging window information over said first virtual channel of said communications link and said local node and said remote node exchanging graphical data over said second virtual channel of said communications link.

20. The system of claim 20 wherein the first virtual channel and the second virtual channel comprise the same virtual channel.

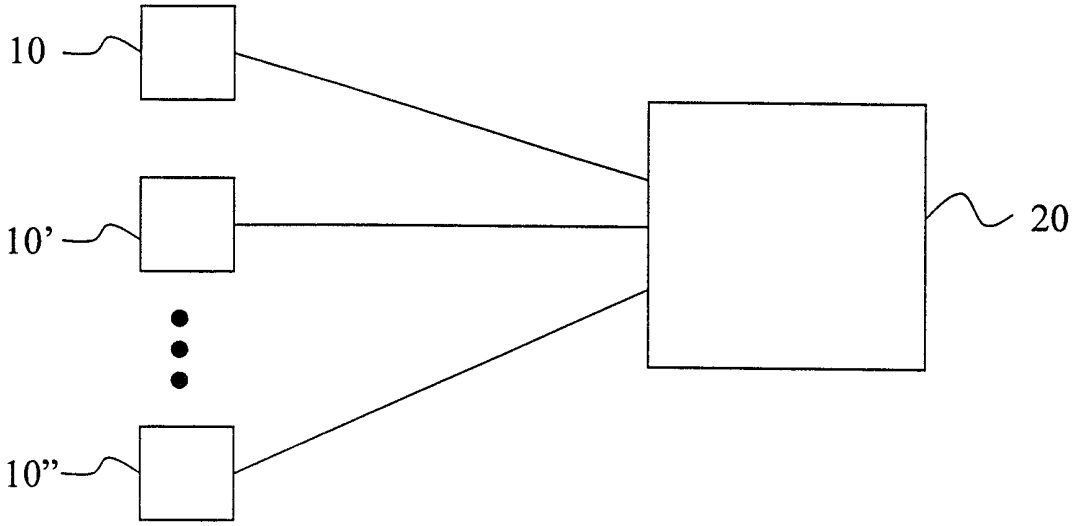


Fig. 1

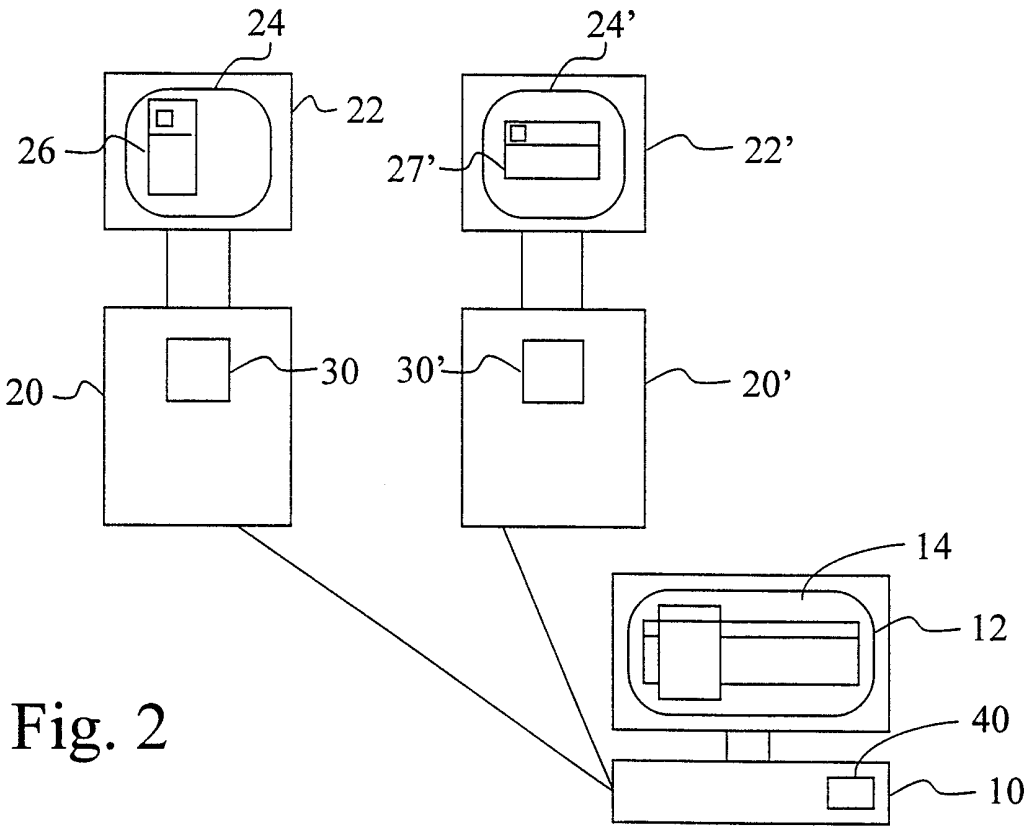


Fig. 2

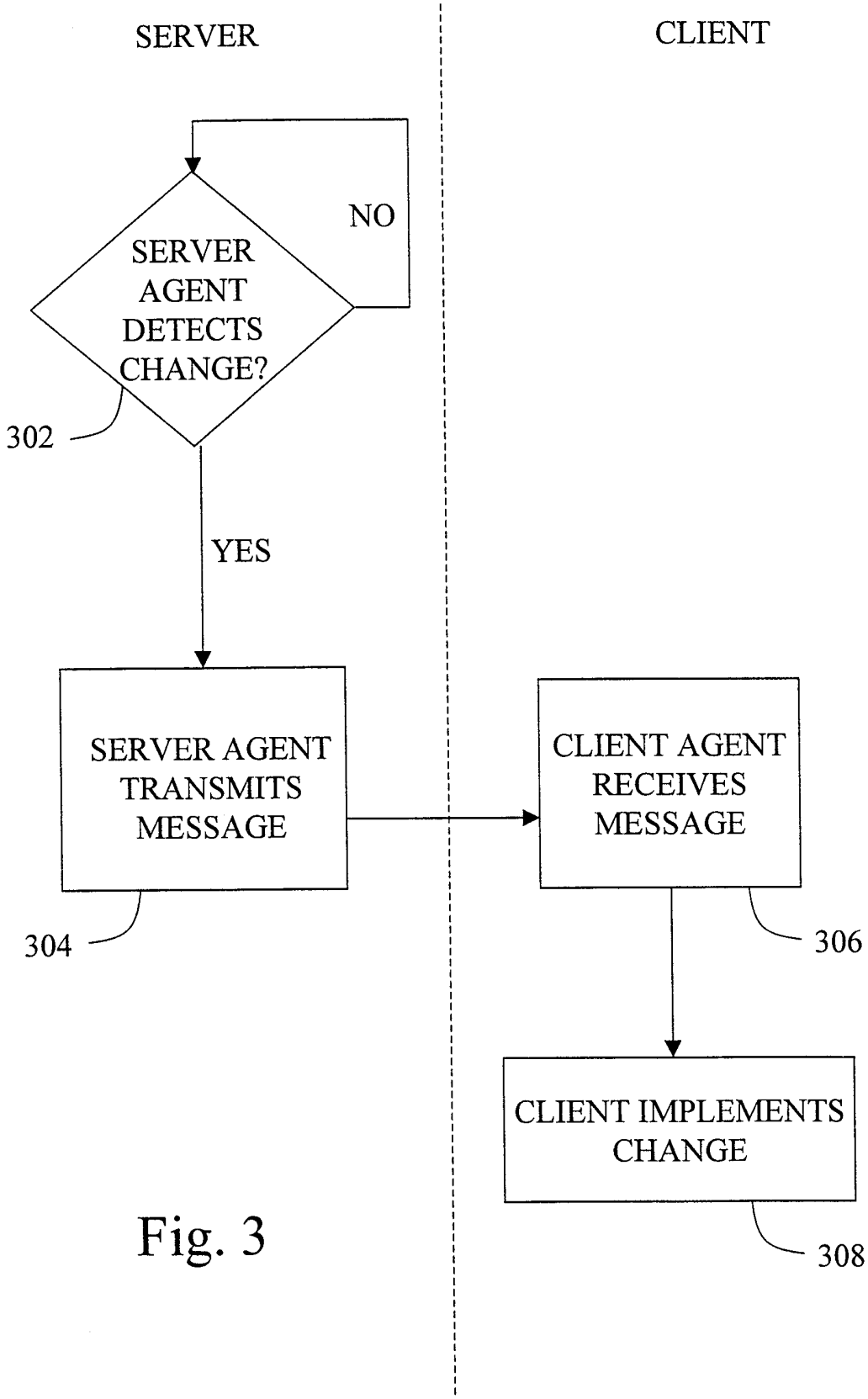
SERVER
CLIENT
SERVER AGENT DETECTS CHANGE?
NO
302
YES
SERVER AGENT TRANSMITS MESSAGE
304
CLIENT AGENT RECEIVES MESSAGE
306
CLIENT IMPLEMENTS CHANGE
308

Fig. 3

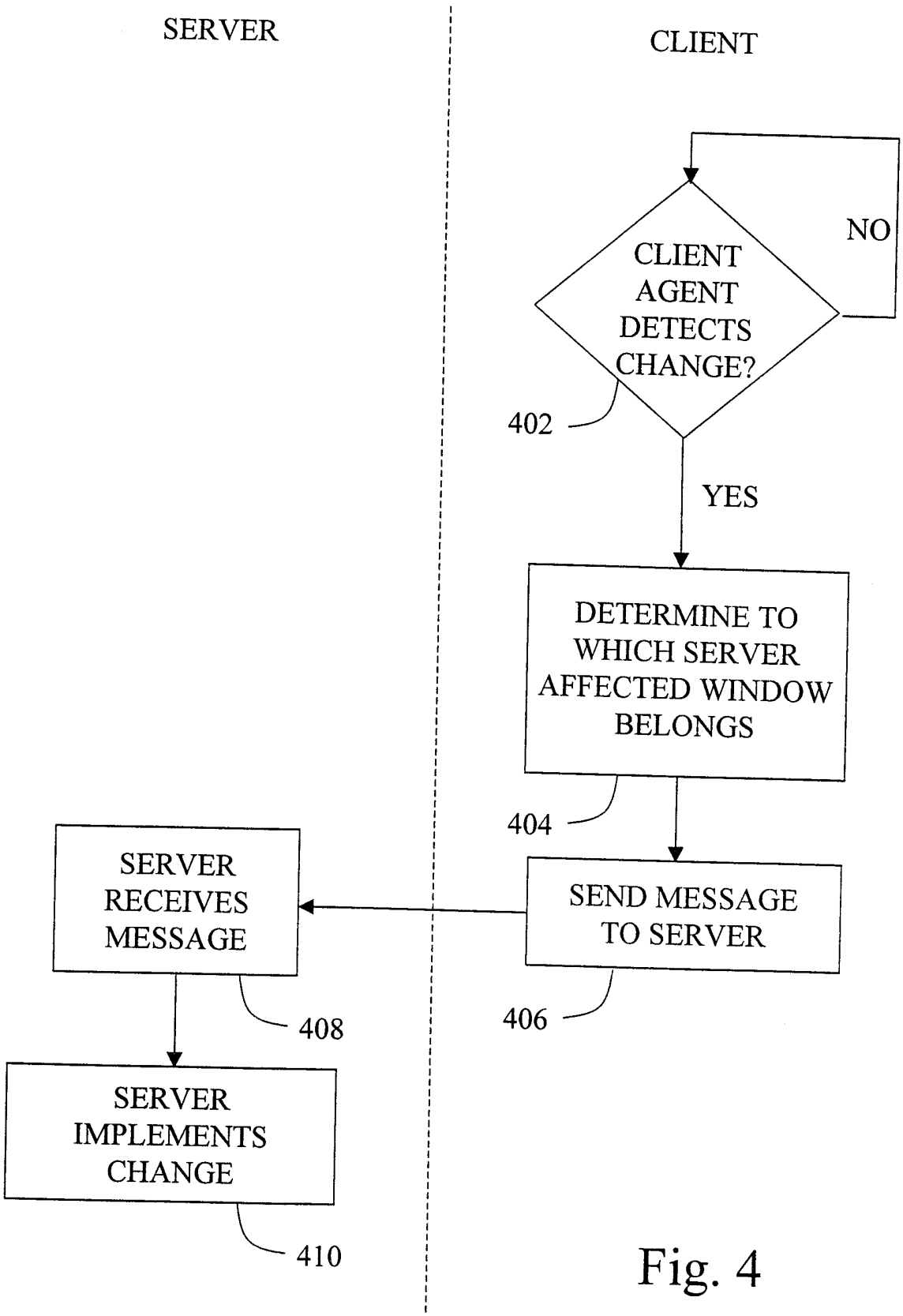
SERVER
CLIENT
CLIENT AGENT DETECTS CHANGE?
NO
402
YES
DETERMINE TO WHICH SERVER AFFECTED WINDOW BELONGS
404
SEND MESSAGE TO SERVER
406
SERVER RECEIVES MESSAGE
408
SERVER IMPLEMENTS CHANGE
410

Fig. 4

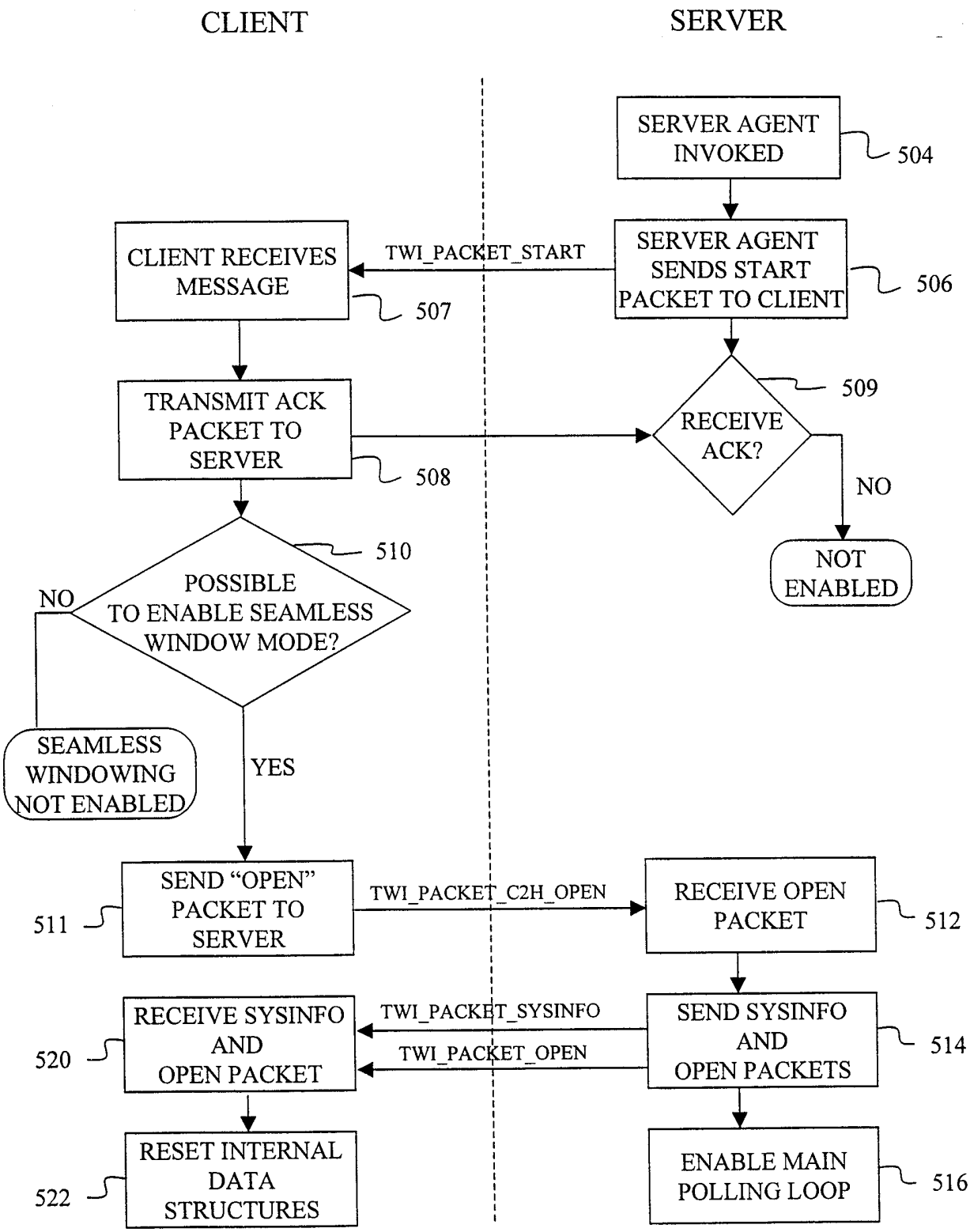
CLIENT
SERVER
SERVER AGENT INVOKED
504
SERVER AGENT SENDS START PACKET TO CLIENT
506
TWI_PACKET_START
CLIENT RECEIVES MESSAGE
507
TRANSMIT ACK PACKET TO SERVER
508
509
RECEIVE ACK?
NO
NOT ENABLED
510
POSSIBLE TO ENABLE SEAMLESS WINDOW MODE?
NO
SEAMLESS WINDOWING NOT ENABLED
YES
SEND "OPEN" PACKET TO SERVER
511
TWI_PACKET_C2H_OPEN
RECEIVE OPEN PACKET
512
SEND SYSINFO AND OPEN PACKETS
514
TWI_PACKET_SYSINFO
TWI_PACKET_OPEN
RECEIVE SYSINFO AND OPEN PACKET
520
RESET INTERNAL DATA STRUCTURES
522
ENABLE MAIN POLLING LOOP
516

Fig. 5

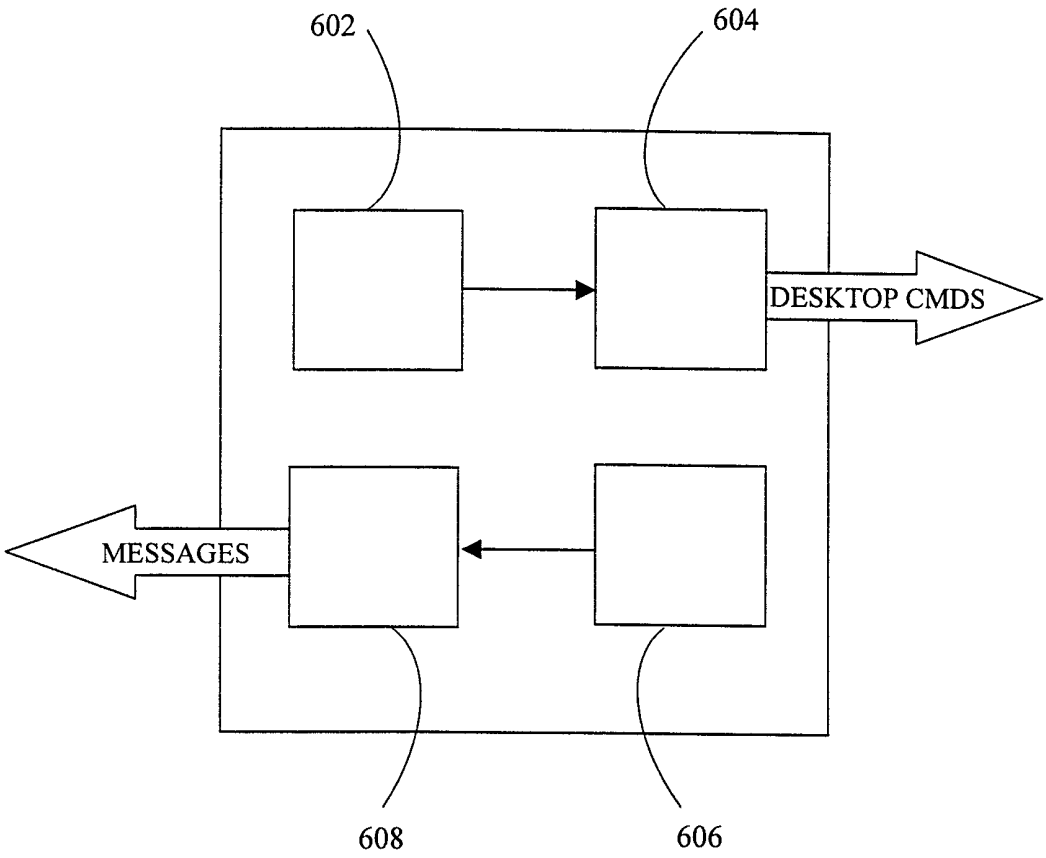


Fig. 6

# INTERNATIONAL SEARCH REPORT

Ir ıational Application No
PCT/US 99/11534

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F9/44 G06F9/46

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 6 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)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

| Category ° | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| X | US 5 537 548 A (FIN TONG-HAING ET AL)<br>16 July 1996 (1996-07-16)<br>the whole document | 1-20 |
| A | ABE T ET AL: "DISTRIBUTED COOPERATIVE CONTROL FOR SHARING APPLICATIONS BASED ON THE MERMAID MULTIPARTY AND MULTIMEDIA DESKTOP CONFERENCING SYSTEM"<br>NEC RESEARCH AND DEVELOPMENT,<br>vol. 34, no. 1,<br>1 January 1993 (1993-01-01), pages 122-131, XP000363016<br>ISSN: 0547-051X | 1-19 |
| A | WO 97 28623 A (GOLAN GILAD ;ZANGVIL ARNON (IL); ZANGVIL AVNER (IL); MENTA SOFTWAR)<br>7 August 1997 (1997-08-07)<br>the whole document | 1-19 |

☐ Further documents are listed in the continuation of box C.

☒ 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 | Date of mailing of the international search report |
|---|---|
| 9 September 1999 | 16/09/1999 |

| Name and mailing address of the ISA | Authorized officer |
|---|---|
| European Patent Office, P.B. 5818 Patentlaan 2<br>NL – 2280 HV Rijswijk<br>Tel. (+31–70) 340–2040, Tx. 31 651 epo nl,<br>Fax: (+31–70) 340–3016 | Fonderson, A |

1

Form PCT/ISA/210 (second sheet) (July 1992)

# INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 99/11534

| Patent document cited in search report | Publication date | Patent family member(s) | Publication date |
|---|---|---|---|
| US 5537548 A | 16-07-1996 | JP 5046568 A | 26-02-1993 |
| | | CA 2071451 A,C | 09-02-1993 |
| | | EP 0527590 A | 17-02-1993 |
| WO 9728623 A | 07-08-1997 | IL 116804 A | 06-12-1998 |
| | | AU 1397097 A | 22-08-1997 |