

(12) PATENT
(19) AUSTRALIAN PATENT OFFICE

(11) Application No. AU 199739638 B2
(10) Patent No. 730088

(54) Title
A system, method and article of manufacture for seamless, server application support of network and non-network client terminals

(51)⁷ International Patent Classification(s)
H04L 029/06

(21) Application No: **199739638**

(22) Application Date: **1997.08.06**

(87) WIPO No: **WO98/06207**

(30) Priority Data

(31) Number	(32) Date	(33) Country
08/692489	1996.08.06	US

(43) Publication Date : **1998.02.25**

(43) Publication Journal Date : **1998.04.30**

(44) Accepted Journal Date : **2001.02.22**

(71) Applicant(s)
Verifone, Inc.

(72) Inventor(s)
Richard Cowan; Gordon P. Eckley; Prasad V. R. Panchangam; Winston C. W. Leong

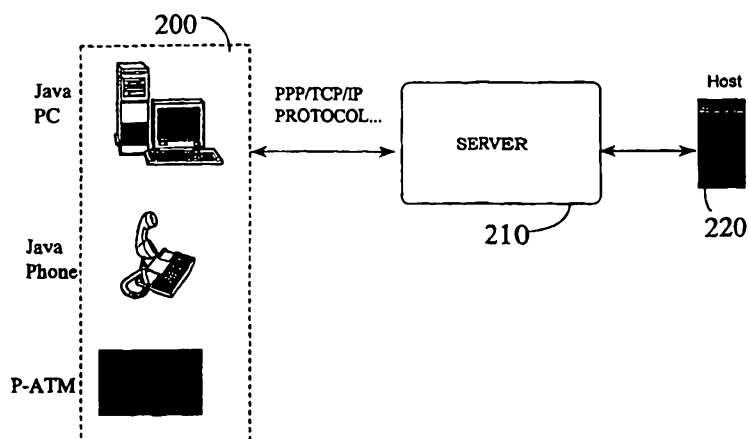
(74) Agent/Attorney
PATENT ATTORNEY SERVICES,26 Ellingworth Parade,BOX HILL VIC 3128

(56) Related Art
EP 666681
EP 326699
US 5506832



(51) International Patent Classification 6 : H04L 29/06		A1	(11) International Publication Number: WO 98/06207
			(43) International Publication Date: 12 February 1998 (12.02.98)
(21) International Application Number: PCT/US97/13057 (22) International Filing Date: 6 August 1997 (06.08.97) (30) Priority Data: 08/692,489 6 August 1996 (06.08.96) US (71) Applicant (for all designated States except US): VERIFONE, INC. [US/US]; Suite 400, Three Lagoon Drive, Redwood City, CA 94065 (US). (72) Inventors; and (75) Inventors/Applicants (for US only): COWAN, Richard [US/US]; 281 Portlock Road, Honolulu, HI 96825 (US). ECKLEY, Gordon, P. [US/US]; 5925 Happy Pines Drive, Foresthill, CA 95631 (US). PANCHANGAM, Prasad, V., R. [IN/US]; 2715 South Norfolk Street #205, San Mateo, CA 94403 (US). LEONG, Winston, C., W. [US/US]; 1751 E. Roseville Parkway #1433, Roseville, CA 95661 (US). (74) Agents: STEPHENS, L., Keith et al.; Warren, Perez & Stephens, Suite 710, 8411 Preston Road, Dallas, TX 75225 (US).		(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, HU, IL, 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, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG). Published With international search report. Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.	

(54) Title: A SYSTEM, METHOD AND ARTICLE OF MANUFACTURE FOR SEAMLESS, SERVER APPLICATION SUPPORT OF NETWORK AND NON-NETWORK CLIENT TERMINALS



(57) Abstract

A plurality of clients (200) are connected to one or more servers (210). When a client initiates a connection with a server (210), the server (210) responds to the request for connection by transmitting a message back to the client to determine whether the client is a network terminal or not. The client responds with a message that is received by an application dispatcher at the server (210) which takes one of a pair of actions based on whether the client is a network terminal. If the client terminal is a network terminal, then the application dispatcher spawns a server application in the server which responds to the client application in the client. Going forward, the server application responds to all future requests from the client application. If the client is not a network terminal, then the application dispatcher initiates a client application in the server (210) to service the client terminal application requirements. Requests from the client application on behalf of the client terminal are subsequently serviced by a server application at the server (210) which communicates to the client terminal via the client application at the server (210).

**A SYSTEM, METHOD AND ARTICLE OF MANUFACTURE
FOR SEAMLESS, SERVER APPLICATION SUPPORT OF
NETWORK AND NON-NETWORK CLIENT TERMINALS**

5

COPYRIGHT NOTIFICATION

Portions of this patent application contain materials that are subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document, or the patent disclosure, as it appears in the Patent and
10 Trademark Office.

Field of the Invention

This invention generally relates to improvements in computer systems, and more particularly, to system software for managing a
15 network of heterogeneous client terminals communicating with a server in a consistent manner.

Background of the Invention

Recently, it has become increasingly fashionable to speak of
20 "intelligent," "smart," or "programmable" terminals and systems. Very few mainframe or peripheral manufacturers omit such a device from their standard product line. Although "intelligence," like beauty or art, is in the eye of the beholder, the adjective generally connotes that the device has a degree of autonomy or
25 processing ability which allows it to perform certain tasks without assistance from the mainframe to which it is connected. Many

such devices are programmable by virtue of including a microprocessor.

While operational devices are somewhat hazy and non-standard, a
5 device is referred to as a terminal if a user interacts with the device to communicate to a host processor, referred to as a server in a network computing environment. Examples of terminals include keyboard/printer terminals, cathode-ray tube (CRT) terminals, remote-batch terminals, real-time data-acquisition and control
10 terminals, transaction and point-of-sale terminals, and smart terminals.

A terminal is considered to be intelligent if it contains, hard-, firm-, and or software which allows it to perform alphanumeric or
15 graphic message entry, display buffering, verifying, editing and block transmissions, either on host or human command. If the terminal contains a microprocessor which runs a standard program to service the terminal, and not arbitrary, user-loaded programs, the terminal has a fixed function, and is still just an
20 intelligent terminal. Only when the device contains a general purpose computer which is easily accessible to the ordinary user for offering a wide range of programs selectable by a user or by devices attached to the device does the terminal become a network terminal in accordance with a preferred embodiment.

25

Sun has recently introduced a new language that is designed to provide consistency for network applications, named Java. Java is a general-purpose, concurrent, class-based, object-oriented programming language and support structure, specifically

5 designed to have as

few implementation dependencies as possible. Java allows application developers to write a program once and then be able to run it everywhere on a computer network.

10 The Java language solves many of the client-side problems by:

- o enabling dynamic class bindings;
- o providing enhanced portability of applications; and
- o providing a secure environment in which applications execute.

15

Java is compiled into bytecodes in an intermediate form instead of machine code (like C, C++, Fortran, etc.). The bytecodes execute on any machine with a Java bytecode interpreter. Thus, Java applications can run on a variety of client machines, and the

20 bytecodes are compact and designed to transmit efficiently over a network which enhances a preferred embodiment with universal clients and server-centric policies.

With Java, developers can create robust User Interface (UI)

25 components. Custom "widgets" (e.g. real-time stock tickers, animated icons, etc.) can be created, and client-side performance

is improved. Unlike HTML, Java supports the notion of client-side validation, offloading appropriate processing onto the client for improved performance. Dynamic, real-time applications can be created using the above-mentioned components.

5

Sun's Java language has emerged as an industry-recognized language for "programming the Internet." Sun defines Java as: "a simple, object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high-performance, multithreaded, dynamic, buzzword-compliant, general-purpose programming language. Java supports programming for the Internet in the form of platform-independent Java applets." Java applets are small, specialized applications that comply with Sun's Java Application Programming Interface (API) allowing developers to add "interactive content" to Web documents (e.g. simple animations, page adornments, basic games, etc.). Applets execute within a Java-compatible browser (e.g. Netscape Navigator) by copying code from the server to client. From a language standpoint, Java's core feature set is based on C++. Sun's Java literature states that Java is basically "C++, with extensions from Objective C for more dynamic method resolution".

A network terminal in accordance with a preferred embodiment would execute Java applications in stand-alone mode, but have the capability to interact with a server for such functions as retrieving information, database processing, massive computation processing

and access to shared devices such as high-speed printers, plotters and magnetic tapes.

The term "distributed computing" refers both to the devices at
5 remote locations and to the logic which has been used to enhance the intelligence of the devices. Such distributed or decentralized computing with remote intelligent terminals and network terminals is a fact of life in today's computer literate society.

10 There are a number of drawbacks to distributed computing environments which are not found in a centralized computing environment. First, hardware problems: when a user locates a software solution that is optimal for the user's terminal environment, the software often will not execute on the host
15 processor that is universally accessible by other's in a company. Moreover, the software will often be incompatible with other user's terminals.

Second, interfacing problems: a nonstandard terminal might
20 require a special-purpose interface and might not be recognized by the host. Even standard interfaces are notorious for crashing the operating system. In any case, "mixed systems" containing multiple vendor hardware are becoming the norm, but lead to the blame for system problems being placed on the other system, and
25 result in difficult debugging and resolving of system problems.

Third, host operating system support for a heterogeneous terminal environment can be a nightmare. To provide support for all of the various protocols, communication rates and processing demands
5 with the peculiarities intrinsic to a motley crew of downstream terminals is a system administration headache.

Fourth, local software support: this type of support ranges from minimal (say, a compiler for the particular terminal) to a mail
10 program that is compatible with every different terminal attached to the host server. Some applications can be rebuilt for a particular terminal by simply recompiling the application, but many are only distributed as runtime modules with no support provided for some terminals.

SUMMARY OF THE INVENTION

In accordance with the invention, there is provided a distributed computer system including a client terminal and a server which communicate via a network, including:

- (a) the client terminal initiating connection to the server utilizing the network;
- (b) the server responding to the initial connection by transmitting an enquiry message to the client terminal;
- (c) the client terminal responding to the enquiry message with a message including identification information indicative of the client terminal being a network terminal or a non-network terminal and identifying a client application the client terminal requires;



(d) the server receiving and analyzing the identification information to determine if the client terminal is a network terminal or a non-network terminal; and

(d1) if the client terminal is a network terminal, then the client loader on the server starts the client application, if necessary, on the client terminal utilizing the network and
5 starts a server application to service future requests from the client terminal; and

(d2) if the client terminal is a non-network terminal, then the server initiates the client application and server application on the server for processing the client application at the server for the client terminal.

The invention also provides a method for distributing computing between a server and
10 a client terminal which communicate via a network, including the steps of:

(a) initiating connection of the client terminal to the server utilizing the network;

(b) responding to the initial connection request at the server by transmitting an enquiry message to the client terminal;

(c) responding to the enquiry message at the client terminal with a message
15 including identification information indicative of the client terminal being a network terminal or a non-network terminal and identifying a client application the client terminal requires;

(d) receiving and analyzing the identification information at the server to determine if the client terminal is a network terminal or a non-network terminal; and

20 (d1) loading a server application if the client terminal is a network terminal, which starts the client application if necessary, on the client terminal utilizing the network and starts a server application to service future requests from the client terminal; and

(d2) loading a server application on the server, if necessary, which initiates a client



application on the server for processing the client application at the server on behalf of the client terminal, if the client terminal is a non-network terminal.

Also the invention provides a computer program embodied on a computer-readable medium for enabling a distributed computing system, including a client terminal and a

5 server which communicate via a network, including:

(a) a code segment for initiating connection of the client terminal to the server utilizing the network;

(b) a code segment for responding to the initial connection request at the server by transmitting an enquiry message to the client terminal;

10 (c) a code segment for responding to the enquiry message at the client terminal with a message including identification information indicative of the client terminal being a network terminal or a non-network terminal and identifying a client application the client terminal requires;

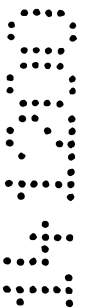
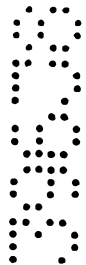
(d) a code segment for receiving and analyzing the identification information at
15 the server computer to determine if the client terminal is a network terminal or a non-network terminal; and

(d1) a code segment for loading a server application if the client terminal is a network terminal, which starts the client application if necessary, on the client terminal utilizing the network and starts a server application to service future requests from the client
20 terminal; and

(d2) a code segment for loading a server application, if necessary, on the server which initiates the client application on the server for processing the client application at the server on behalf of the client terminal, if the client terminal is a non-network terminal.



The foregoing problems are overcome in an illustrative embodiment of the invention in a network computing environment in which a plurality of clients are connected to one or more servers. When a client initiates a connection with a server, the server responds to
5 the request for connection by transmitting a message back to the client to determine whether the client is a network terminal or not. The client responds with a message that is received by an application dispatcher at the server which takes one of a pair of actions based on whether the client is a network terminal. If the
10 client terminal is a network terminal, then the application



dispatcher spawns a server application in the server which responds to the client application in the client. Going forward, the server application responds to all future requests from the client application. If the client is not a network terminal, then the
5 application dispatcher initiates a client application in the server to service the client terminal application requirements. Requests from the client application on behalf of the client terminal are subsequently serviced by a server application at the server which communicates to the client terminal via the client application at
10 the server.

Brief Description of the Drawings

The above and further advantages of the invention may be better
15 understood by referring to the following description in conjunction with the accompanying drawings, in which:

Figure 1 is a block schematic diagram of a computer system for example, a personal computer system on which the inventive
20 object oriented information manager operates;

Figure 2 illustrates a client - server network in accordance with a preferred embodiment;

25 Figure 3 illustrates a server architecture in accordance with a preferred embodiment;

Figure 4 illustrates a client - server architecture in accordance with a preferred embodiment;

- 5 Figure 5 illustrates a first client request to a server in accordance with a preferred embodiment;

Figure 6 illustrates a client server environment which accesses support services in accordance with a preferred embodiment;

10

Figure 7 is an architecture diagram of a client - server system in accordance with a preferred embodiment;

- Figure 8 is an architecture diagram of a client - server system in
15 accordance with a preferred embodiment;

Figure 9 is an architecture diagram of a client - server system in accordance with a preferred embodiment;

- 20 Figure 10 illustrates the message format utilized in accordance with a preferred embodiment;

- Figure 11 presents a table showing additional details associated with the device types, commands and data blocks in accordance
25 with a preferred embodiment;

Figure 12 presents additional detail on the message format in accordance with a preferred embodiment;

Figure 13 illustrates the display commands and responses in
5 accordance with a preferred embodiment;

Figure 14 presents the status values associated with various operations in accordance with a preferred embodiment; and

10 Figure 15 is a communication flow diagram in accordance with a preferred embodiment.

Detailed Description

The invention is preferably practiced in the context of an operating
15 system resident on a computer such as a SUN, IBM, HP, or a Windows NT computer. A representative hardware environment is depicted in Figure 1, which illustrates a typical hardware configuration of a computer 100 in accordance with the subject invention. The computer 100 is controlled by a central processing
20 unit 102 (which may be a conventional microprocessor) and a number of other units, all interconnected via a system bus 108, are provided to accomplish specific tasks. Although a particular computer may only have some of the units illustrated in Figure 1, or may have additional components not shown, most server
25 computers will include at least the units shown.

Specifically, computer **100** shown in Figure **1** includes a random access memory (RAM) **106** for temporary storage of information, a read only memory (ROM) **104** for permanent storage of the computer's configuration and basic operating commands and an
5 input/output (I/O) adapter **110** for connecting peripheral or network devices such as a disk unit **113** and printer **114** to the bus **108**, via cables **115** or peripheral bus **112**, respectively. A user interface adapter **116** is also provided for connecting input devices, such as a keyboard **120**, and other known interface
10 devices including mice, speakers and microphones to the bus **108**. Visual output is provided by a display adapter **118** which connects the bus **108** to a display device **122**, such as a video monitor. The computer has resident thereon and is controlled and coordinated by operating system software such as the SUN Solaris, Windows
15 NT or JavaOS operating system.

Figure **2** illustrates a client-server network in accordance with a preferred embodiment. A set of consumer devices (client terminals **200**) are attached to a server **210** and the server is attached to a
20 legacy host **220** to process applications requiring information at the host **220**. The connection could be by means of the Internet, a dialup link, token ring, cellular phone, satellite, T1 or X.25 telco link or other communication means.

Server Software

25 The sever software is written using a combination of Java, C or possibly C++. C or C++ will be used mainly to implement platform

dependent code (such as dealing with the comm ports). While a preferred embodiment discloses support for a dial up network and Internet processing utilizing TCP/IP, one of ordinary skill in the art will readily realize that a token ring, SNA or other network, such as
5 those discussed in US Patents (5,530,961; 5,491,796; 5,457,797; 5,442,791; 5,430,863; 5,394,401; 5,291,597; 5,287,537; 5,287,461; 5,201,049; 4,991,089; and 4,588,211) could be readily interchanged as the network.

10

Architecture

A server architecture in accordance with a preferred embodiment supports two types of client terminals.

Network terminals. These are client terminals capable of directly
15 executing the Java applications on the client terminal which are initially stored on a server. The server will simply download this code to the client's network terminal which the client will then execute to provide a particular service. This service may or may not interact with other clients or servers. Network terminals can be
20 connected to a server through a dial up modem link, directly through a local area network, or by other network communication means in accordance with a preferred embodiment.

Non-network terminals. These are client's terminals which are
25 not capable of executing Java applications on the client terminal. When dealing with this class of client the server will execute the

application on behalf of the client. In this case the server will only expect necessary input and output operations to be performed by the client terminal. An example of how to connect a plurality of non-network terminals to a host server is described in US Patent
5 5,287,461, the disclosure of which is hereby incorporated by reference in its entirety.

Figure 3 illustrates a server architecture in accordance with a preferred embodiment. A client **300** would initiate a connection
10 with a server **350** by, for example, dialing in to a modem pool which is intercepted by the point-to-point stack software **311** which conforms information received to the TCP layer **312** which obtains a socket **313** for connecting the client **310** to the server **350**. The Java net layer **314** further refines the request to conform
15 with the TERMIO and NET layer **315** which passes the request along to the application dispatcher **319**. The application dispatcher **319** spawns the appropriate server application selected from the server applications **330**. On a non-network terminal, The non-network terminal initiates a "first connection" by dialing up a
20 modem, for example. The dial up goes through the native OS **316** (Solaris or Windows NT dial up layer) and is connected with the serial communication in the VFI.SERIAL layer **317** which abstracts the serial input/output functions into a higher level communication layer. The VFI.NET layer **315** takes the abstracted
25 serial layer and maps it into a similar communication as the communication from the network terminal **300**. It makes the

dialup asynchronous connection appear to the server application as a new socket connection.

Network Terminal - "First Connection"

- 5 Figure 4 illustrates a client - server architecture in accordance with a preferred embodiment. The architecture is illustrated initially for a network terminal for clarity and then follows with a non-network terminal. Processing commences at **400** when a network terminal requests connection through a layered
- 10 communication system to a set of server threads **420** which are triggered by a detection of a "ring" **430** to initiate possible client updates and the subsequent client application to server application processing. "Ring" refers to a "first connection" in socket processing in accordance with a preferred embodiment.
- 15
- The network terminal makes its connection through the Point-to-Point-Protocol stack **411** utilizing the TCP layer **412** and the sockets layer **413**, which is like an electrical socket, for attaching terminals to communication sockets to facilitate communication
- 20 through the network. All of this is managed by the Java.net **414** which connects the socket **1111** via the TCP layer **412** and the PPP stack **411**. The layer above is the VFI.net and VFI.TERMIO **415** which is responsible for detecting that the connection is made and mapping the connection to an application dispatcher **431** to
- 25 further process the first connection (ring) request.

The server **450** waits for a "first connection" request much like an interrupt manager. When a "first connection" request arrives, then the application dispatcher has a method that detects a connect
5 request or a LAN "first connection" request that would arrive through the TCP layer as a socket connect. That connection is translated into a logical ring which is equivalent to an event or interrupt. The server **450** responds to the "first connection" with a query initiated by the application dispatcher **431** requesting "who
10 are you" via an enquiry message asking for identification by the client loader thread **421**. The network terminal responds with ID information, including the identification of the application that the network terminal requires. If the terminal answers with an identifier indicating that the terminal is a network terminal, then
15 the client loader thread **421** performs any necessary client application updates via a download using a file transfer program such as UDP or FTP, or any other socket layer protocols that are available for network file transfers to the network terminal **400**.

20

Network Terminal - First Client Request to Server

Figure 5 illustrates a first client request to a server in accordance
25 with a preferred embodiment. When a first client request is transmitted from the network terminal **500** with a client

application resident thereon **510** to the server **550**, the application dispatcher **530** spawns the corresponding server application **520** for servicing the request at the server **550** via the assigned socket **1112**. The server application **520** responds to the request and
5 transmits information to the network terminal **500**. The application dispatcher **530** has completed its responsibilities for this client **500** and can return to a wait state until the next "first connection" request from a client. The client application request could be as simple as a get current time request or a request for
10 data from a server database.

Network Terminal - Subsequent Client Request to Server

Figure **6** illustrates a network terminal **600** with a downloaded client application **610** which accesses support services in the
15 server **650** through its assigned server application **620** in accordance with a preferred embodiment. The terminal **600** communicates to a server application **620** which accesses host processing capabilities and database services **640** to service requests emanating from the client application **610**. The server
20 application **620** handles any events that originate from the client application **610** via the assigned socket **1112**. These events could include data requests from a database application, or data transfer to a server. Remote data from another server application could also be accessed by the client. Server application **620** accesses
25 support services directly or via a socket interface **660**.

Non-network Terminal - "First Connection"

Figure 7 is an architecture diagram of a client - server system in accordance with a preferred embodiment. A layered

5 communication system **700** is used by a non-network terminal **710** to detect a ring providing an indicia of communication **740** and dispatch an application **730**. Dispatching an application **730** also initiates a server thread **720** for servicing the client request. The non-network terminal **710** initiates a "first connection" by
10 dialing up a modem, for example. The dial up goes through the native OS **711** (Solaris or Windows NT dial up layer) and is connected with the serial communication in the VFI.SERIAL layer **712** which abstracts the serial input/output functions into a higher level communication layer. The VFI.NET layer **715** takes
15 the abstracted serial layer and maps it into a similar communication as the communication from the network terminal. It makes the dialup asynchronous connection appear to the server application as a new socket connection **1111**. The communication is an event **740** that triggers actions by the application dispatcher
20 **741** which responds to the "first connection" event by requesting ID information from the client, via an enquiry message, and starting the requested client application **720** at the server **750**.

25 **Non-network Terminal - First Client Request to Server**

Figure 8 is an architecture diagram of a client - server system in accordance with a preferred embodiment. The client application **822** is responsible for managing the non-network terminal **810**. The client application **822** writes information, utilizing a server version of VFI.TERMIO **855**, to and responds to key presses by the non-network terminal **810** at the server **850**. The client application **822** initially makes a request for service from a socket **1112** that is associated with the non-network terminal **810** when the application dispatcher **840** spawns the client application **822**.

10

When the first request **845** is generated by the client application **822** residing on the server **850**, at application startup, the first request for service is routed in the server **850** to the application dispatcher **840** and spawns the server application **820** which will handle subsequent requests. The server application **820** makes a request for service from a socket **1112** that is associated with the client application **822** which transmits an appropriate command through the VFI.TERMIO **855** to the VFI.SERIAL layer **856** using the operating system communication support **857** to the non-network terminal **810**. This processing is identical to the network terminal processing with the exception that all applications reside on the server **850** as opposed to a Java application executing remotely on the network terminal.

15

20

25

One advantage of Java is that it is machine independent and does not care whether a Java application resides on the client or the

server. In the case of the non-network terminal, the client application resides in the server and controls the java incapable terminal.

5 **Non-network Terminal - Subsequent Client Requests to Server**

Figure 9 is an architecture diagram of a client - server system in accordance with a preferred embodiment. A layered communication system 900 is used by a non-network terminal 910 to manage the interconnection of a server Application 940 to a client application 920 and facilitate communication between the terminal 910 and server application 940 via a client application 920 resident on the server 950. Figure 9 shows the processing after the first request has been completed and the client application 920 is coupled with the server application 940 via the assigned socket 1112 just as in the network terminal example, except the client application 920 and server application 940 both reside on the server 950.

If a terminal responds with a message that indicates it is a non-network terminal, then the terminal is supported with the command streams described in Figures 10-14. If the terminal is a network terminal, then the application is downloaded via a FTP or other network file transfer procedure.

25 Figure 10 illustrates the structure of a packet in accordance with a preferred embodiment. Figure 11 shows the format of each field of

a communication and describes the contents of the same. For example, the header is two bytes in length and has various values that correspond to different types of transactions. Similarly, the Packet Type, Header CRC, Sequence #, Data Block and CRC-16
5 fields are described in the table set forth in Figure 11.

Figure 12 represents a table showing additional details associated with the device types, commands and data parameters. For example, the device type field is one byte long and specifies the
10 selected Input/Output device. Figure 13 illustrates the display commands in accordance with a preferred embodiment. The display's device type is zero. Figure 14 presents the status values associated with various requested operations in accordance with a preferred embodiment.

15

Figure 15 is a communication flow diagram in accordance with a preferred embodiment. A terminal 1500 either has firmware or an application 1504 that initiates a connection 1506 with a server 1502 by contacting a dispatcher 1508. The connect initiate 1506
20 also connects a socket 1111 to handle the connection. The dispatcher 1508 transmits an identification enquiry 1510 which the client terminal replies to with an identification message 1512. In the case of a network terminal, the client loader 1522 performs any necessary client application updates 1520 on the client
25 terminal 1500. In the case of a non-network terminal, the dispatcher starts the client application. The client then sends a

request to start the server application **1530** to the server which results in the connection of a socket **1112** and the server application **1550** being started and a confirmation message **1532** being transmitted back to the client application **1540**. Then, when
5 the client application **1540** requests data **1542** from the server application **1550**, the server application **1550** responds with the application response data **1560**.

Application Dispatcher - Control Flow

Application Dispatcher startup

Configured modem ports that will take part in transactions are pre-configured. The Application Dispatcher (AD) startup code

5 looks at this configuration stream to determine the number of S threads (serial port listeners). S classes instantiate a VFI.NET.serversocket object which in turn create a VFI.NET.ModemIO.ModemPort object. The ModemPort object binds to a low level VFI.NET.ModemIO.Port object which utilizes

10 native methods to configure and wait on the communications port.

```

S0
{
    serversocket SOSocket = new serversocket ("socket1111", 1);
15 // Listener object
    {
        socket SOConnSocket= SOSocket.accept(); //
    }
    Translates to
        WaitDevice(CONNECT)
20         ReadAndValidate (RequestID);
        return RequestID, SOConnSocket;
    }
}

```

25 **Request Processing**

- As illustrated above, S threads are transient threads. And even when alive they perform efficient waits (No CPU cycles are consumed). The AD receives the RequestID from each S thread. Request processing is performed by database lookup. Typically
- 5 Requests, are simple text messages with delimiters and are parsed using a StringTokenizer object.

```
StringTokenizer stParseHelp = new StringTokenizer ((String)
Request);
10
field1 = stParseHelp.nextToken();
field2 = .... and so on.
```

- The AD will query a database to determine which applications
- 15 should be initiated based on the enquiry message utilizing an SQL query of the form:

```
"SELECT <Field ClassPath> from <TableName> where <f1 = field1
and .....>;
```

- 20 is handled by the JDBC layers to return data to the AD. The AD is now ready to run the client thread.

```
ClientThread = new Thread (field1, field2... , S0ConnSocket);
```

The field list contains appropriate fields (those required for client application processing) and are passed down to the client thread along with the connected socket object.

5

Client Threads

Client Threads proxy the actual application. Application output meant for the terminal's devices are routed out using VFI.TERMIO as directives to the client terminal's firmware. The connected socket (which translates to a live dial-up connection) is passed
10 down from the AD to the client thread. Client threads are long living - usually transferring data to corresponding servlets that initiate connections to upstream hosts or make database transactions. Despite the fact that client threads can be JDBC aware, servlets handle database transactions. This helps to
15 maintain code constancy when the same client class is downloaded to a Java capable terminal for remote execution.

Terminal I/O is performed through a VFI.TermIO object that in turn instantiates a VFI.TermIO.ServProtocol object. The protocol
20 object implements the actual data transfer with the client terminal. The protocol object requires the socket object passed down from the AD to the client thread.

CO (Appropriate Request fields, S0ConnSocket)

25 {

VFI.TermIO IOObject = new TermIO (S0ConnSocket); //IO
object //instantiation. This cascades into a ServProtocol
Object instantiation.

5 IOObject.WriteString (StringIndex); //Displays a particular
string on the P-ATM.

//If the client needs to retrieve data from upstream hosts
(OmniHost, VISA etc), //or needs data from a database it makes
10 a TCP stream connection to a servlet.

//This is consistent with the behavior of the network
terminal which would //make the same connection over PPP.

clientTransObject = new Socket (<Host>, <Well known
socket>);

15 // Explained further down under initial client requests

..... //Further processing

// Send out host requests

20 clientTransObject.write (HostRequest);

clientTransObject.read (HostResponse);

IOObject.WriteString (StringIndex + n); //Displays status on
the P-ATM.

}

25

Initial Client Request processing

The AD runs a T thread (spawned off during startup) that listens on a well-known socket (e.g. 1112) waiting for initial

- 5 ClientRequests from a client application. The T thread processes the ClientRequest to determine which servlet class needs loading.

T

{

10 ClientInitialRequestListener = new ServerSocket (<wellknown
socket (e.g. 1112)>);

// Wait for initial requests and spawn off server

connSocket = ClientInitialRequestListener.accept();

15

connSocket.Stream.read (InitialRequest);

Parse (InitialRequest);

HostThread H0 = new Thread (connSocket, "class name");

20 }

The T thread is a daemon thread and lives as long as the AD lives. When the client application is downloaded to a Java capable terminal initial requests arrive over the PPP link.

25

Host Threads or Servlets

Host Threads (H) service client requests for upstream and database connectivity. A host thread can make TCP connections with
 5 remote hosts, forward financial transactions originating from the client application and route the response.

```
H0 (connSocket)
{
10     connSocket.Stream.read (ClientRequest);
    ParseRequest (StringTokenizer);

    Socket upstreamSock = new Socket (upstreamHost, Port);

15     //Transact

    connSocket.Stream.Write (HostResponse);
}
```

Transient and Long-living Threads in the Application

20 **Dispatcher**

A sockets based abstraction of the Win32 Communication API

Consistence in the access of transport layer services needs no over emphasis. The design of the PTS server aims to provide a uniform
 25 interface to third party client component and server component applet writers to the async dial-up protocol module and the

system's TCP/SLIP/PPP stack. This interface comprises a set of Java Classes collectively called VFI.NET.*. It should be noted that this package does not provide pure TCP/UDP/IP specific objects and methods that are already defined and implemented in

- 5 java.net.*. Programmers, however, do not need to explicitly import java.net.*. This is automatically done by the package. Further, this document does not discuss the functionality of java.net.* which may be found in the appropriate JDK documentation. It, merely, details a class design that overloads methods specifically
- 10 necessary to build a BSD sockets like layer between calling applets (servlets) and the machine specific Java serial communications package.

Hierarchy

- A uniform upper edge interface for the ModemIO classes permits
- 15 easy replacement of the implementation. The actual modem handling code, for instance, may use the TAPI client calls instead of direct Win32 communication calls. Multiple libraries that conform to the same interface allow different link level protocol stacks (like MNP3). This ensures the constancy (and hence direct
- 20 portability) of VFI.ModemIO.*.

Required ModemIO Functionality

1. Open an end-to-end async, duplex dial-up connection. The station address (InetAddress as in TCP/IP) is the dial string.
- 25 Configure upon connection.

2. Listen for an incoming dial-up connection. The listen port (analogous to the bound TCP port) is the COM port. In this regard the valid port numbers range from 0 - 0xFF (which is the maximum number of COM ports allowed in NT). Configure
5 upon initialization.
3. Obtain Input and Output streams that re-direct from/to the open connection.
- 10 4. Hang-up (close as in TCP/IP) a live connection.

The following classes form a part of VFI.ModemIO.* :-

```
15          Raw Serial Port Handling
public class VFI.ModemIO.Port
{
    //Constructors
    public Port (int nPortNum);
20    public Port (int nPortNum, int nBaud, int nParity, int
nDataBits, int nStopBits);
    public Port (int nPortNum, String sCfgStr);
    public Port (String sPortName);
    public Port (String sPortName, String sCfgStr);
25
```

```
//Methods
public void close();
public int getPortID();
public String getPortName();
5 public String getCfgStr();
public InputStream getInputStream();
public OutputStream getOutputStream();
}

10 Modem initialization and methods
public class VFI.ModemIO.ModemPort
{
    //Constructors
    public ModemPort (int nPortNum);
15 public ModemPort (Port objPort);
    public ModemPort (String sPortName);
    public ModemPort (int nPortNum, String sInitString);
    public ModemPort (Port objPort, String sInitString);
    public ModemPort (String sPortName, String sInitString);
20
    //Methods
    public Port getPort();
    public boolean connect (String sDialString);
    public void disconnect();
25 public void reset();
    public boolean configure (String sCfgStr);
```

```
        public boolean configureDM (String sCfgStr);  
  
    }
```

- 5 Programmers must use `getPort()` to capture a stream and transfer data over the `ModemPort`. `Configure(String)` sends out an AT command and returns `TRUE` if the modem returned `OK<cr><lf>`. `configureDM(String)` sends out the same command to the modem when in data mode.

10

NET - The Sockets wrapper

- The package encapsulates two major classes found in `java.net.*` - `Socket` and `ServerSocket`. To present a familiar interface and yet avoid conflicts, the package instantiates its own socket and
15 `serversocket` objects via constructors that take an extra parameter (that identifies the lower object that needs to be instantiated). This is illustrated after the class definition.

Station address resolution

- 20 The `InetAddress` object refers to an unique long value that corresponds to the machines TCP/IP address. The async dial-up line may however use multiple COM ports to open a connection with the host. Heuristically, it may seem that fitting the TCP/IP host/machine address into the native COM support library will
25 permit overloading of `InetAddress` and hence enhance elegance. This, however, results in extra and avoidable complexity. In this

regard, InetAddress will still correspond only to a TCP/IP address. The versions of the java.net.Socket constructor that accept the host name (as a String) will, instead, be overloaded. This value will now refer to a dial String that identifies the remote station address.

5

Socket initialization and connection

```
public class VFI.NET.socket
{
    //Constructors
10    public socket (String sHost, int nPort, int nProtocolType);
    /*    nProtocolType may take one of two values :
           PF_INET    #defined to 1
           PF_VFI_PTS_MODEMIO #defined to 2
           Passing a value of 0 causes the use of
15    java.net.Socket.* /

    //Methods
    public void close();
    public String getStationAddress();
20    public int getPort();
    public InputStream getInputStream();
    public OutputStream getOutputStream();
}

25    public class VFI.NET.serversocket
    {
```

```
//Constructors
public serversocket(int nPort, int nProtocolType);
/*    nProtocolType may take one of two values :
    PF_INET    #defined to 1
5        PF_VFI_PTS_MODEMIO #defined to 2
    Passing a value of 0 causes the use of
    java.net.ServerSocket.*/

//Methods
10    public socket accept();
    public void close();
    public int getPort();
}

15    Interface Library to native Win32 Comm. API methods
    HANDLE OpenDevice (int nDevNum, DCB * pNewDCB);
    void CloseDevice (HANDLE hDevice);
    int WriteDevice (HANDLE hDev, int nBytesToWrite, unsigned char
    * pWriteBuf);
20    int ReadDevice (HANDLE hDev, int nBytesToRead, unsigned char *
    pReadBuf);
    BOOL ConfigureDevice (HANDLE hDev, DCB * pNewDCB);
```

25

While the invention is described in terms of preferred embodiments in a specific system environment, those skilled in the art will recognize that the invention can be practiced, with modification, in other and different hardware and software environments within the
5 spirit and scope of the appended claims.

The claims defining the invention are as follows:

1. A distributed computer system including a client terminal and a server which communicate via a network, including:

(a) the client terminal initiating connection to the server utilizing the network;

5 (b) the server responding to the initial connection by transmitting an enquiry message to the client terminal;

(c) the client terminal responding to the enquiry message with a message including identification information indicative of the client terminal being a network terminal or a non-network terminal and identifying a client application the client terminal
10 requires;

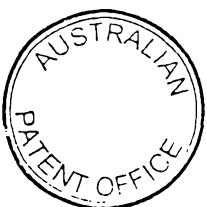
(d) the server receiving and analyzing the identification information to determine if the client terminal is a network terminal or a non-network terminal; and

(d1) if the client terminal is a network terminal, then the client loader on the server starts the client application, if necessary, on the client terminal utilizing the network and
15 starts a server application to service future requests from the client terminal; and

(d2) if the client terminal is a non-network terminal, then the server initiates the client application and server application on the server for processing the client application at the server for the client terminal.

2. The distributed computer system as recited in claim 1, wherein the start of the client
20 application entails a download of the client application to the client terminal.

3. The distributed computer system as recited in claim 1 or 2, in which the client terminal communicates to the server utilizing a dial-up network connection.



4. The distributed computer system as recited in claim 1, 2 or 3 wherein the identification information includes configuration characteristics of the client terminal.
5. The distributed computer system as recited in claim 1, 2, 3 or 4 wherein the network terminal executes Java code on the network terminal.
- 5 6. The distributed computer system as recited in any one of the preceding claims, wherein the same client application is executed on the server and the client terminal.
7. The distributed computer system as recited in any one of the preceding claims, wherein the non-network terminal receives commands from the client application on the server.
- 10 8. The distributed computer system as recited in any one of the preceding claims, including means for passing a client application request to another server to process the request.
9. A method for distributing computing between a server and a client terminal which communicate via a network, including the steps of:
- 15 (a) initiating connection of the client terminal to the server utilizing the network;
- (b) responding to the initial connection request at the server by transmitting an enquiry message to the client terminal;
- (c) responding to the enquiry message at the client terminal with a message including identification information indicative of the client terminal being a network
- 20 terminal or a non-network terminal and identifying a client application the client terminal requires;
- (d) receiving and analyzing the identification information at the server to determine if the client terminal is a network terminal or a non-network terminal; and



(d1) loading a server application if the client terminal is a network terminal, which starts the client application if necessary, on the client terminal utilizing the network and starts a server application to service future requests from the client terminal; and

(d2) loading a server application on the server, if necessary, which initiates a client
5 application on the server for processing the client application at the server on behalf of the client terminal, if the client terminal is a non-network terminal.

10. The method as recited in claim 9, wherein the start of the client application entails a download of the client application to the client terminal.

11. The method as recited in claim 9 or 10, including the step of communicating between
10 the client terminal and the server utilizing a dial-up network connection.

12. The method as recited in claim 9, 10 or 11 wherein the identification information includes configuration characteristics of the client terminal.

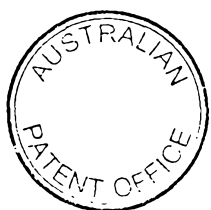
13. The method as recited in claim 9, 10, 11 or 12 wherein the network terminal executes Java code on the network terminal.

15 14. The method as recited in any one of claims 9 to 13, wherein the same client application is executed on the server and the client terminal.

15. The method as recited in any one of claims 9 to 14, wherein the non-network terminal receives commands from the client application on the server.

16. The method as recited in any one of claims 9 to 15, including the step of passing a
20 client application request to another server to process the request.

17. A computer program embodied on a computer-readable medium for enabling a distributed computing system, including a client terminal and a server which communicate via a network, including:



(a) a code segment for initiating connection of the client terminal to the server utilizing the network;

(b) a code segment for responding to the initial connection request at the server by transmitting an enquiry message to the client terminal;

5 (c) a code segment for responding to the enquiry message at the client terminal with a message including identification information indicative of the client terminal being a network terminal or a non-network terminal and identifying a client application the client terminal requires;

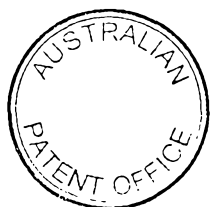
(d) a code segment for receiving and analyzing the identification information at
10 the server computer to determine if the client terminal is a network terminal or a non-network terminal; and

(d1) a code segment for loading a server application if the client terminal is a network terminal, which starts the client application if necessary, on the client terminal utilizing the network and starts a server application to service future requests from the client
15 terminal; and

(d2) a code segment for loading a server application, if necessary, on the server which initiates the client application on the server for processing the client application at the server on behalf of the client terminal, if the client terminal is a non-network terminal.

18. The computer program as recited in claim 17, wherein the start of the client
20 application entails a download of the client application to the client terminal.

19. The computer program as recited in claim 17 or 18, including a code segment for communicating between the client terminal and the server utilizing a dial-up network connection.



20. The computer program as recited in claim 17, 18 or 19, wherein the identification information includes configuration characteristics of the client terminal.

21. The computer program as recited in claim 17, 18, 19 or 20, wherein the network terminal executes Java code on the network terminal.

5 22. The computer program as recited in any one of claims 17 to 21, wherein the same client application is executed on the server and the client terminal.

23. The computer program as recited in any one of claims 17 to 22, wherein the non-network terminal receives commands from the client application on the server.

24. The computer program as recited in any one of claims 17 to 23, including a code
10 segment for passing a client application request to another server to process the request.

25. The computer program as recited in any one of claims 17 to 24, including a code segment for making a dial up connection appear to the server as a socket connection.

26. A distributed computer system substantially as hereinbefore described with reference to the drawings.

15 27. A method for distributing computing substantially as hereinbefore described with reference to the drawings.

28. A computer program for enabling a distributed computing system substantially as hereinbefore described with reference to the drawings.

Dated this 12th day of December 2000

20 PATENT ATTORNEY SERVICES

Attorneys for

VERIFONE, INC



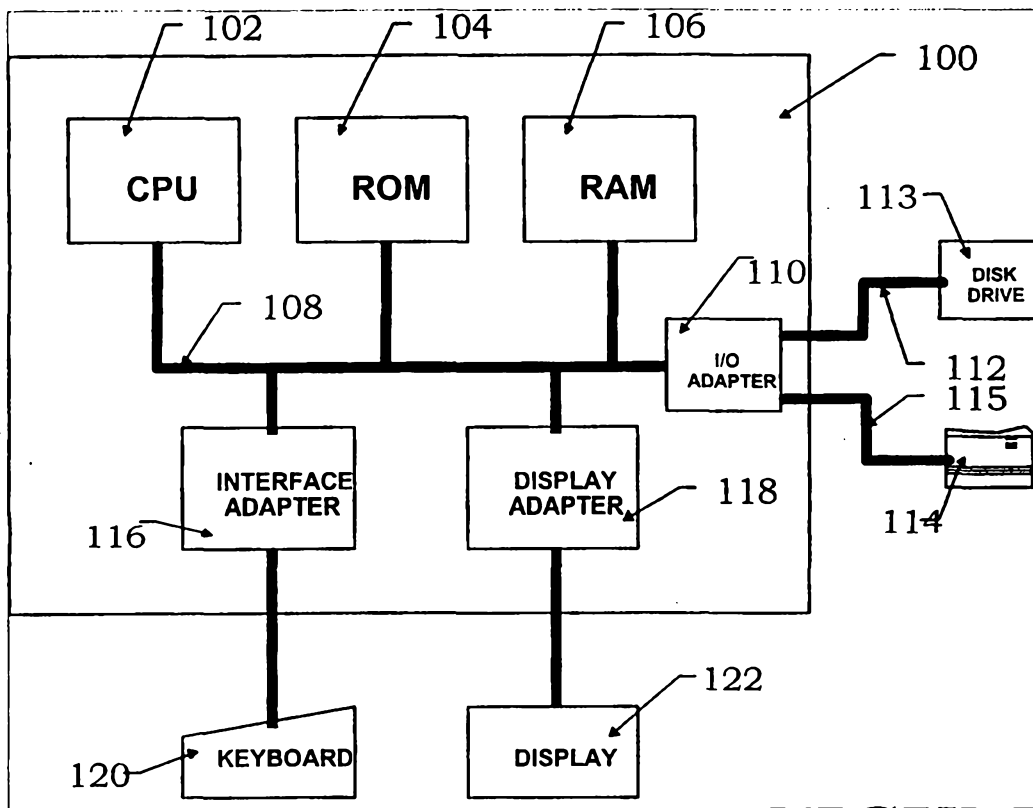


FIGURE 1

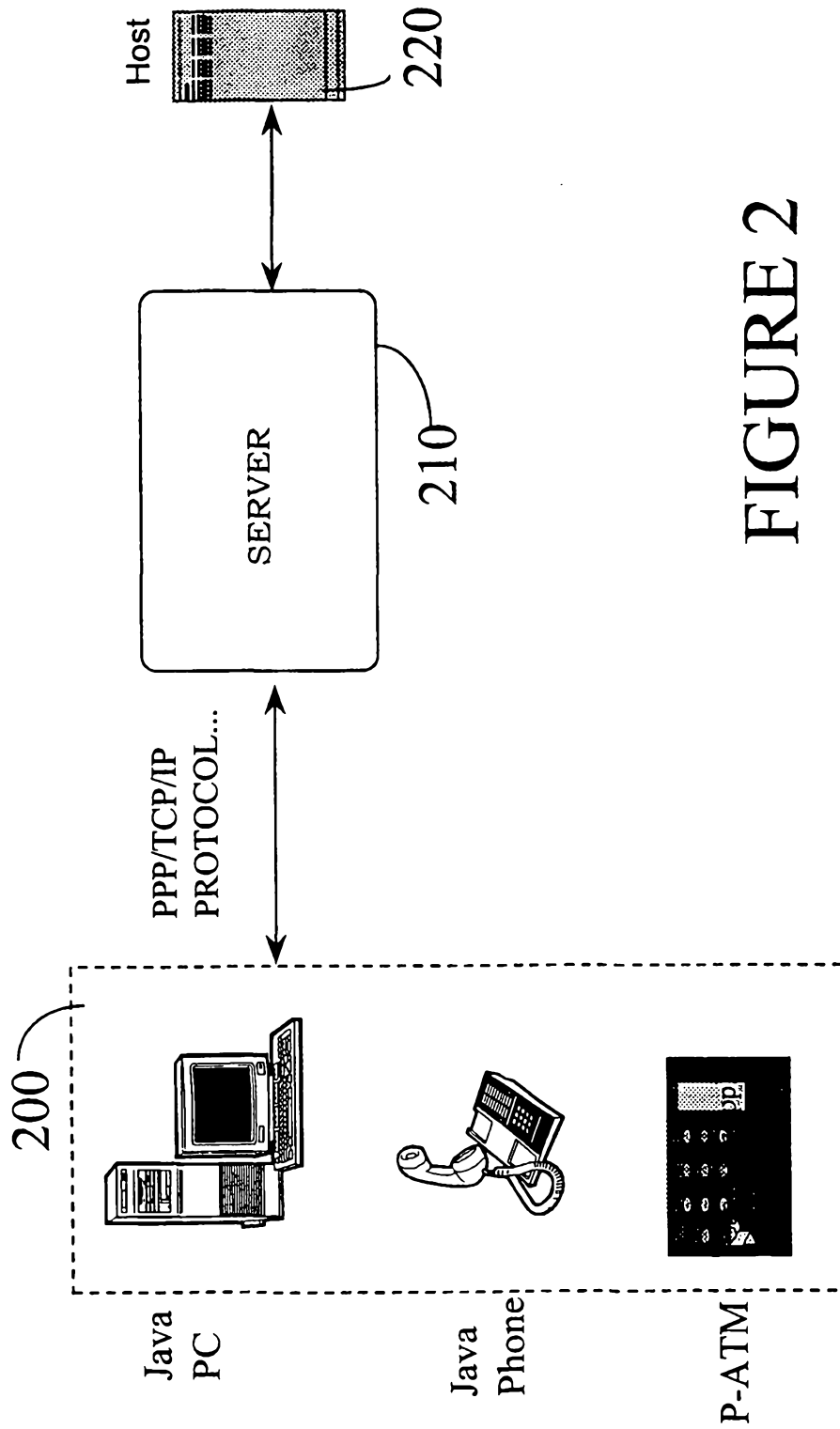
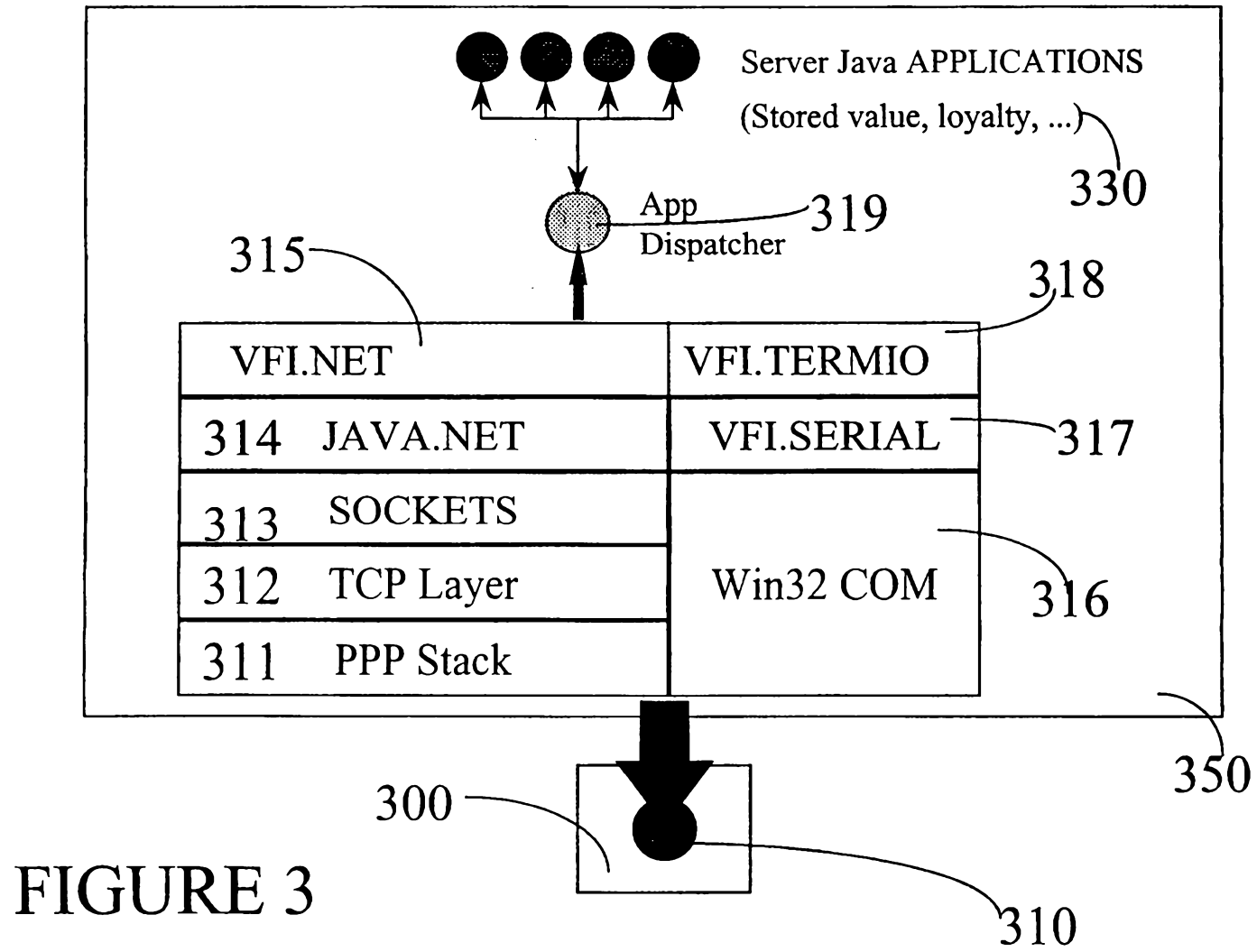


FIGURE 2



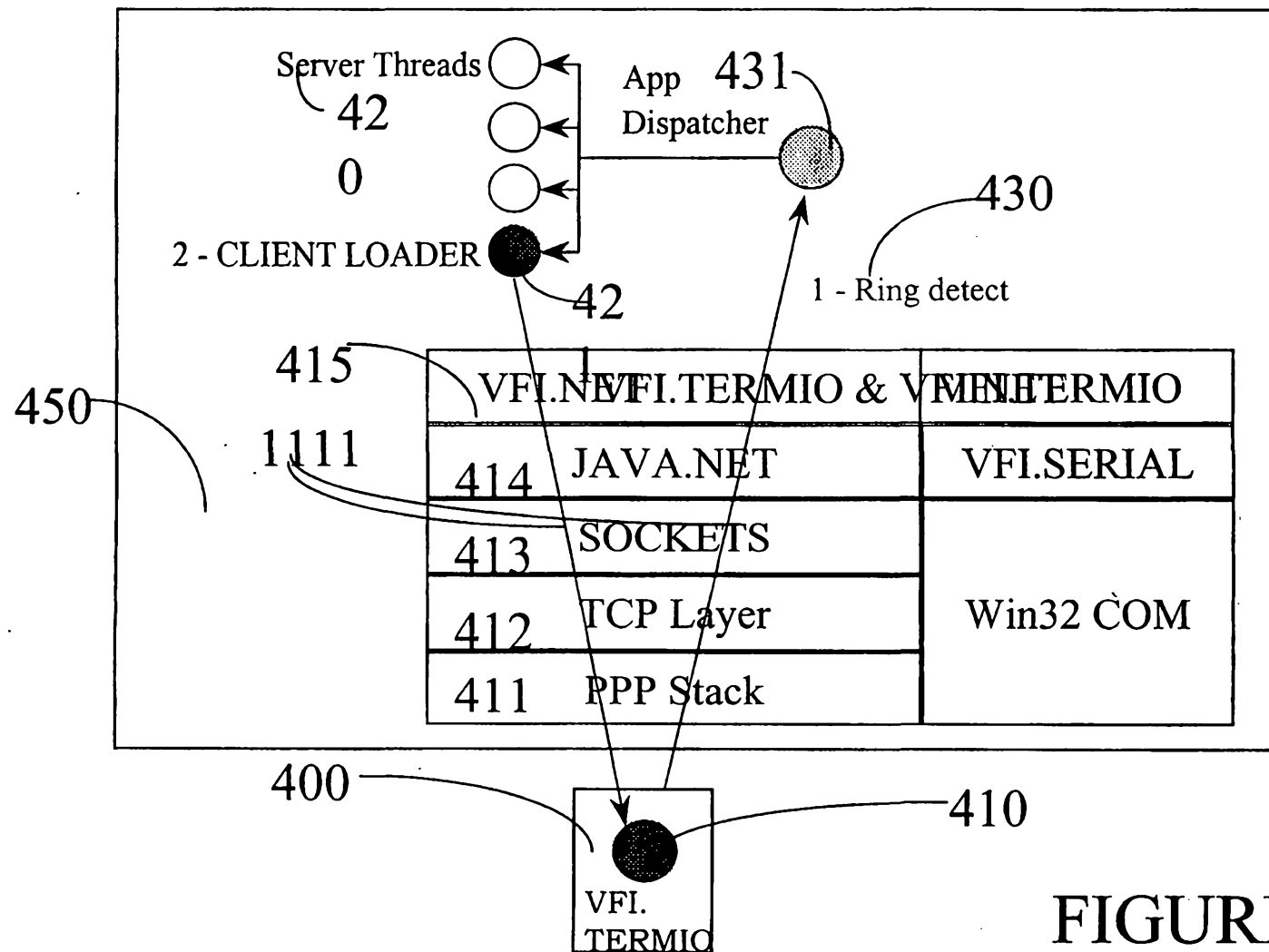


FIGURE 4

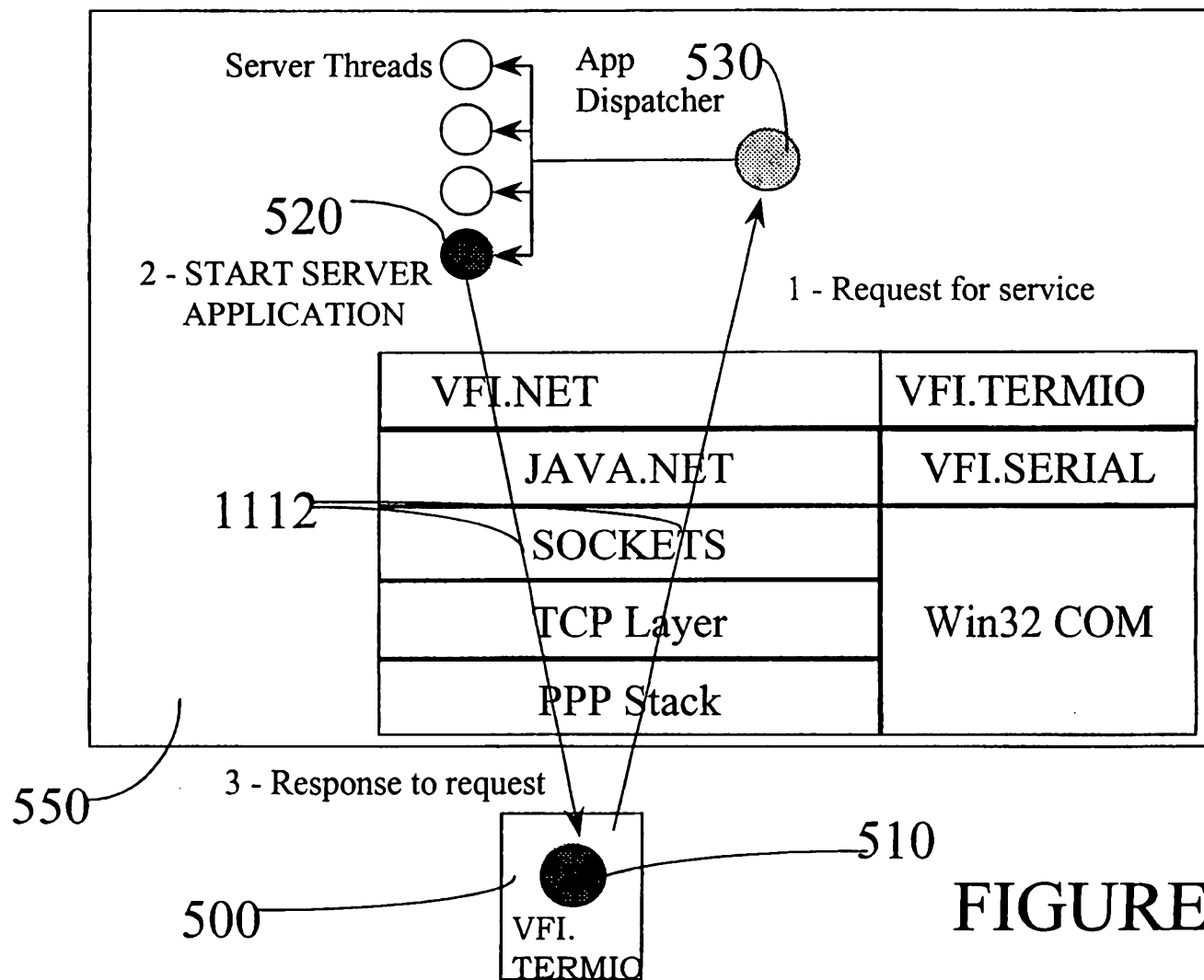


FIGURE 5

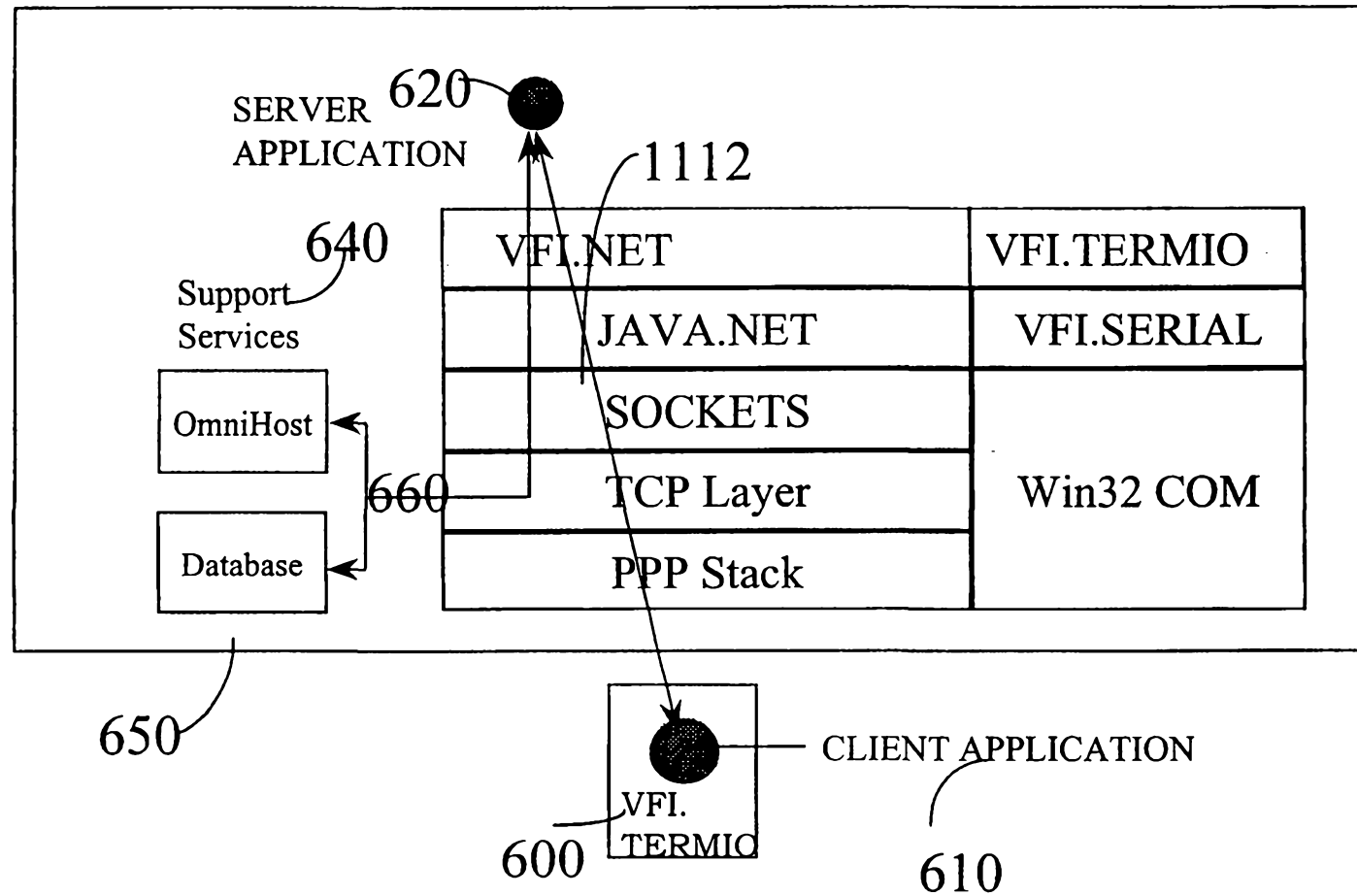


FIGURE 6

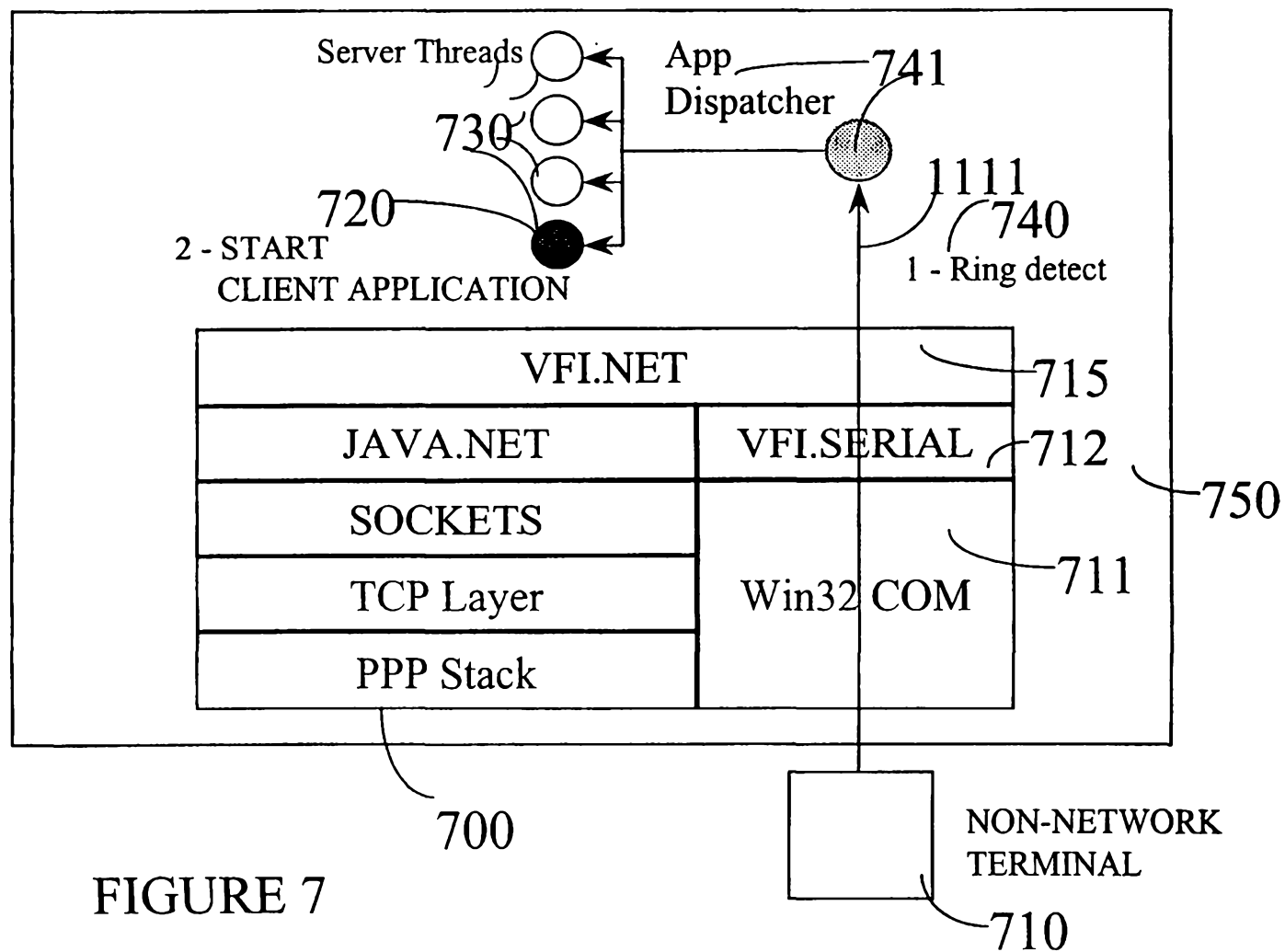


FIGURE 7

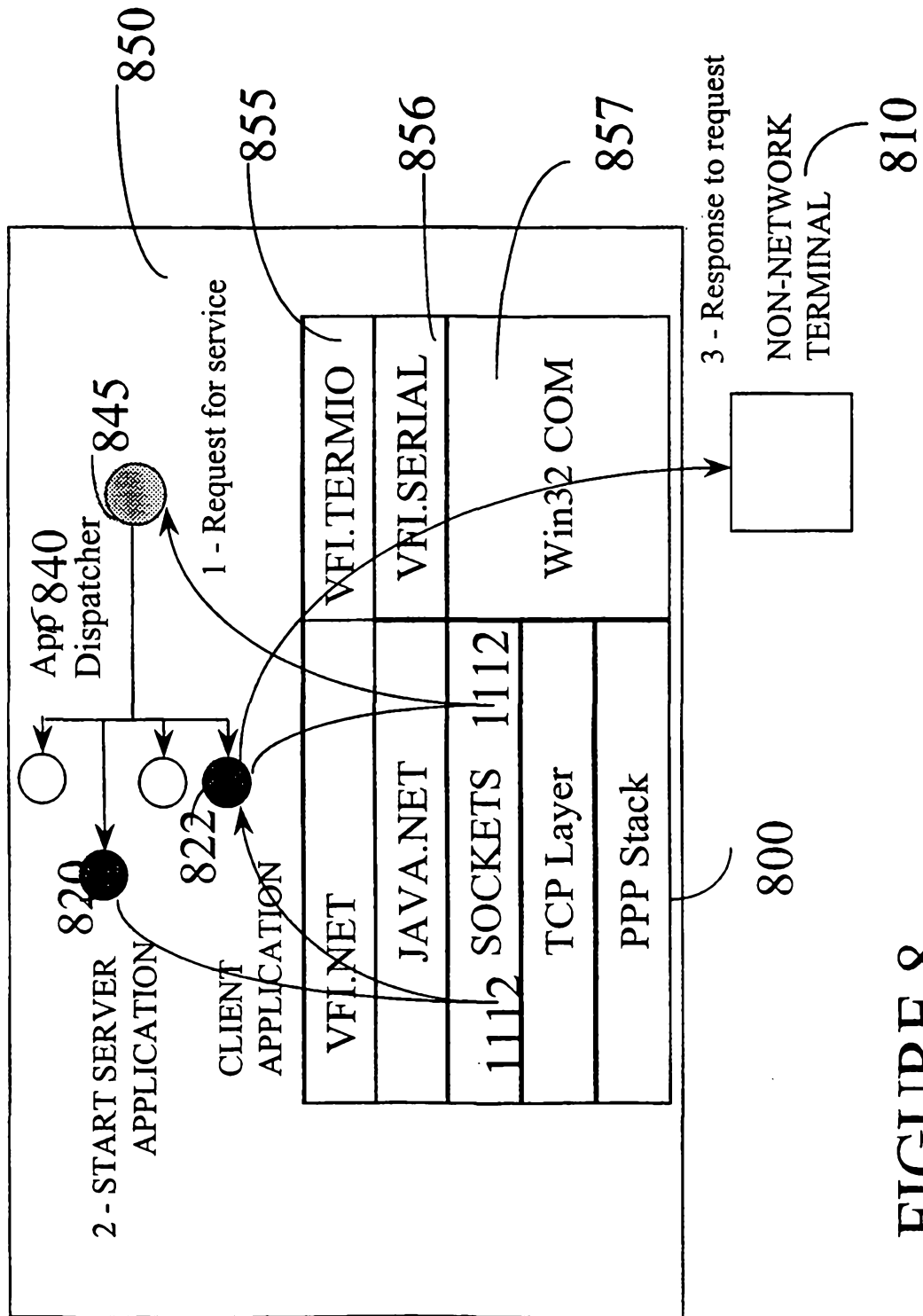


FIGURE 8

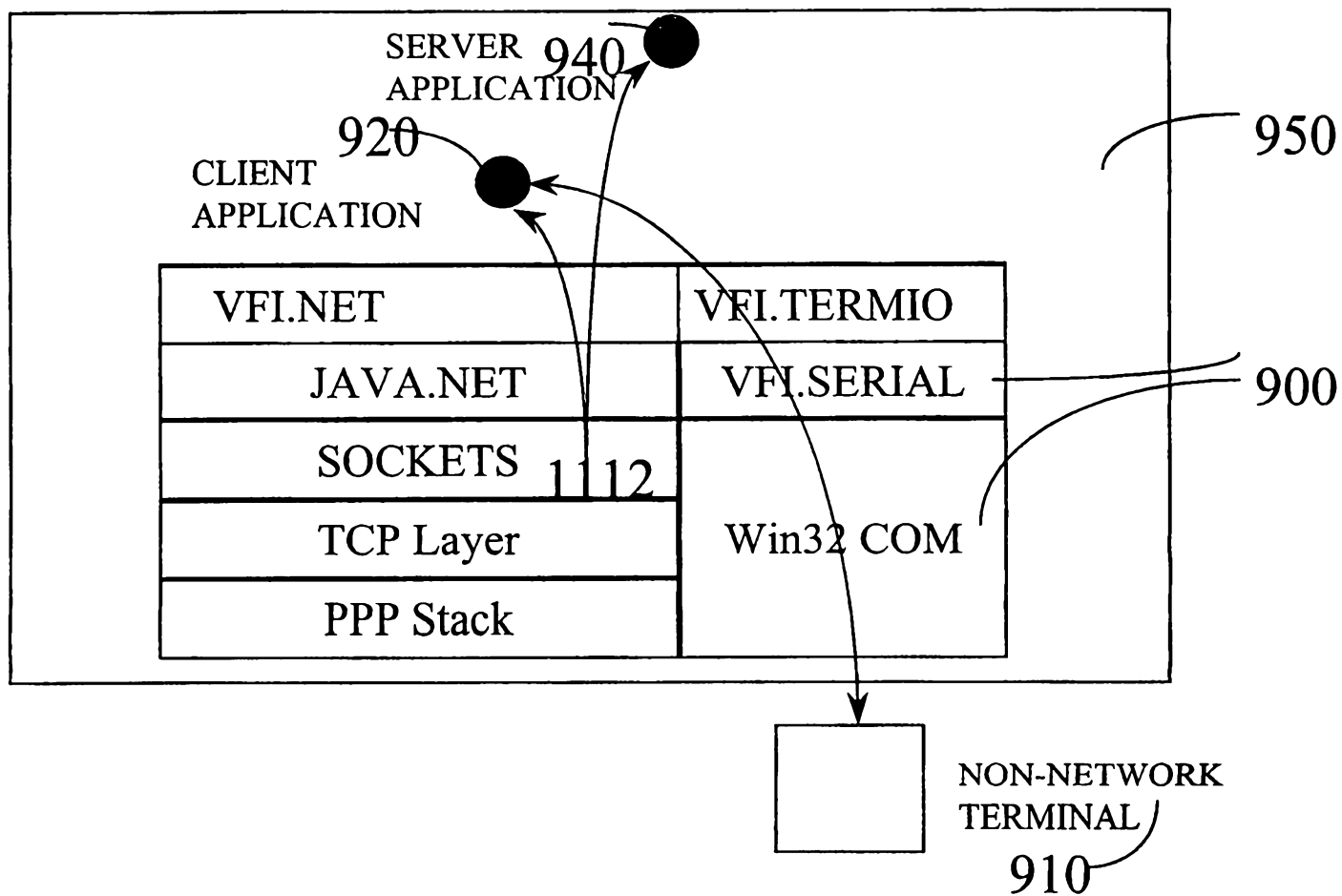


FIGURE 9

ACK/NAK Packet			Sequence #	Data Block	CRC-16
Header	Packet Type/ Byte Count	Header CRC			
Command or Response Packet					

FIGURE 10

Name	Size	Remarks
Header	2 bytes	Has the value 0xAA (MSB), 0x55(LSB)
Packet Type/ Byte Count	2 bytes	The Packet Type is contained in the most 4 significant bits, b15 - b12. This determines if it is a test packet, control packet, or data (command or response) packet.: <u>Control Packet:</u> • ACK • NAK <u>Data (Command or Response) Packet</u> • More Blocks to follow • Last Block • Data Encrypted / Non-encrypted <u>Test Packet:</u> • Server Initiated Test Mode, loopback The Byte Count is contained in the remaining 12 bits, b11 - b0 (4095 max value), the size of the Sequence # and Data Block.
Header CRC	2 bytes	Checksum of the Packet Type/Byte Count. For ACK and NAK packets, this is the last transmitted data.
Sequence #	1 byte	Optional field, valid only for non-ACK/NAK packets. Start block is always 0, subsequent blocks will be incremented by 1.
Data Block	Byte Count - 1	Optional field, valid only for non-ACK/NAK packets. The Command or Response Message may be broken up into smaller packets (blocks), and may further be encrypted.
CRC-16	2 bytes	2 byte CRC calculation, from Sequence # to the last byte of the Data Block field. It is the standard 16-bit CRC-CCITT algorithm: $G(x) = x^{16} + x^{12} + x^5 + 1$

FIGURE 11

Name	Size	Remarks
Length	1 byte	Length of data to follow. (For Block Symmetric Encoding.)
SubDevice Type	1 byte	Selects the subdevice: System, Display, Keypad, and ICC.
Command	1 byte	Command byte is Device Type dependent
Parameters	<variable>	This field is Command and Device Type dependent. For Command Message: command data parameters. For Response Message: 1 byte status, followed by requested data, if any.

FIGURE 12

Command	Description	Data Parameters
0	Store text string into NV memory.	Table, offset, string.
1	Display raw text	String, column.
2	Display preset prompt from NV memory.	Table, offset, column.
3	Set local echo.	None.
4	Clear local echo.	None.
5	Set secure echo (display '*').	None.
6	Clear secure echo.	None.

FIGURE 13

Status	Description
0	Successful operation.
1	Invalid command.
2	Too many characters.
3	Illegal prompt selection.

FIGURE 14

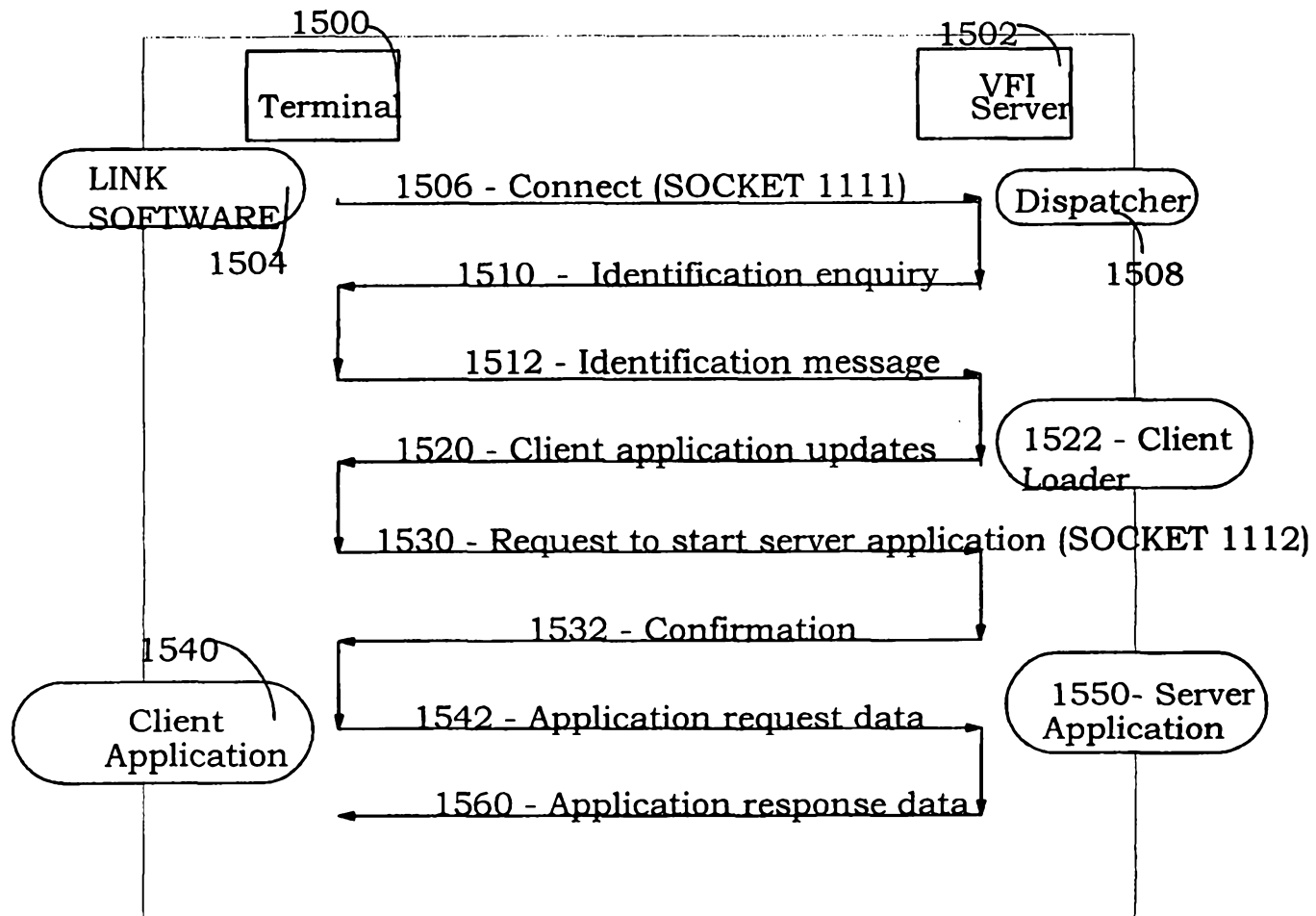


FIGURE 15