PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(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

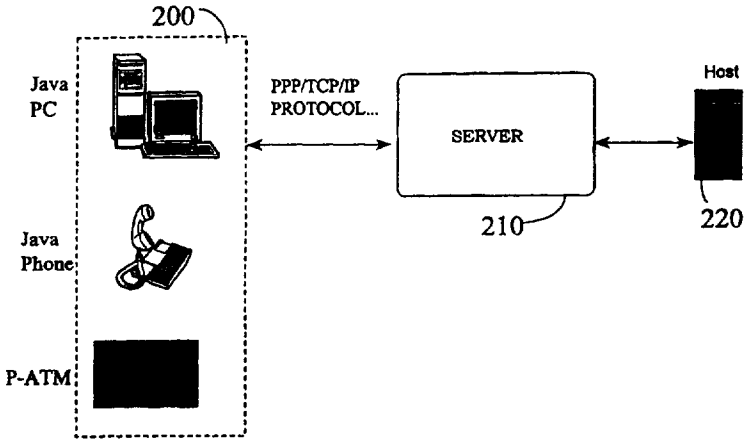
(75) Inventors/Applicants *(for US only)*: COWAN, Richard [US/US]; 281 Portlock Road, Honolulu, HI 96825 (US). ECKLEY, Gordon, P. [US/US]; 5925 Happy Pines Drive, Foresthill, CA 95631 (US). PANCHANGAM, Prasad, V., R. [IN/US]; 2715 South Norfolk Street #205, San Mateo, CA 94403 (US). LEONG, Winston, C., 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).

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		

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

COPYRIGHT NOTIFICATION

Field of the Invention

This invention generally relates to improvements in computer systems, and more particularly, to system software for managing a 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 "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 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 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 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 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 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.

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 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.

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.

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 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) 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.

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 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.

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 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 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 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 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 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

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 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 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 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 the server.

Brief Description of the Drawings

The above and further advantages of the invention may be better 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 object oriented information manager operates;

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

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;

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;

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 accordance with a preferred embodiment;

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

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 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 accordance with a preferred embodiment;

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

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 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 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 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 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 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 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 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

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 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.

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 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 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 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 5,287,461, 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 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 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 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 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"

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 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 appplication to server application processing. "Ring" refers to a "first connection" in socket processing in accordance with a preferred embodiment.

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 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 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 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 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 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**.

Network Terminal - First Client Request to Server

Figure **5** illustrates a first client request to a server in accordance 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 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 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 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 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 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 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 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 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 **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**.

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**.

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 dispacher **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.

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.

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.

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 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 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.

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

```
S0
{
        serversocket S0Socket = new serversocket ("socket1111", 1);
// Listener object
        {
                socket S0ConnSocket= S0Socket.accept(); //
Translates to
         WaitDevice(CONNECT)
                ReadAndValidate (RequestID);
                return RequestID, S0ConnSocket;
        }
}
```

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 Requests, are simple text messages with delimiters and are parsed using a StringTokenizer object.

StringTokenizer stParseHelp = new StringTokenizer ((String) Request);

field1 = stParseHelp.nextToken();
field2 = ... and so on.

The AD will query a database to determine which applications should be initiated based on the enquiry message utilizing an SQL query of the form:
"SELECT <Field ClassPath> from <TableName> where <f1 = field1 and .....>;

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.

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 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 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 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.

```
C0 (Appropriate Request fields,  S0ConnSocket)
{
```

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

    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
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.
    clienTransObject  = new Socket (<Host>, <Well known
socket>);
    // Explained further down under initial client requests

    . . . . .              //Further processing

    // Send out host requests
    clienTransObject.write (HostRequest);
    clienTransObject.read (HostResponse);
    IOObject.WriteString (StringIndex + n); //Displays status on
the P-ATM.
}
```

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

```
T
{
        ClientInitialRequestListener = new ServerSocket (<wellknown
socket (e.g. 1112)>);

        // Wait for initial requests and spawn off server
        connSocket = ClientInitialRequestListener.accept();

        connSocket.Stream.read (InitialRequest);
        Parse (InitialRequest);

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

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.

Host Threads or Servlets

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

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

        Socket upstreamSock = new Socket (upstreamHost, Port);

        //Transact

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

Transient and Long-living Threads in the Application 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 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 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 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 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 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. 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 upon initialization.

3. Obtain Input and Output streams that re-direct from/to the open connection.

4. Hang-up (close as in TCP/IP) a live connection.

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

Raw Serial Port Handling

```
public class VFI.ModemIO.Port
{
        //Contructors
        public Port (int nPortNum);
        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);
```

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

Modem initialization and methods

```
    public class VFI.ModemIO.ModemPort
    {
        //Constructors
        public ModemPort (int nPortNum);
        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);

        //Methods
        public Port getPort();
        public boolean connect (String sDialString);
        public void disconnect();
        public void reset();
        public boolean configure (String sCfgStr);
```

```
public boolean configureDM (String sCfgStr);

}
```

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.

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 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

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 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.

Socket initialization and connection

```
public class VFI.NET.socket
{
        //Constructors
        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
java.net.Socket.*/

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

public class VFI.NET.serversocket
{
```

```
        //Constructors
        public serversocket(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
java.net.ServerSocket.*/

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

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);
int ReadDevice (HANDLE hDev, int nBytesToRead, unsigned char *
pReadBuf);
BOOL ConfigureDevice (HANDLE hDev, DCB * pNewDCB);
```

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 spirit and scope of the appended claims.

CLAIMS

Having thus described our invention, what we claim as new, and desire to secure by Letters Patent is:

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

(a) the client terminal initiating connection to the server computer 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 comprising 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 updates the client application, if necessary, on the client terminal utilizing the network and starts the 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 application at the server for the client terminal.

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

3. The distributed computer system as recited in claim 1, 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, wherein the identification information comprises configuration characteristics of the client terminal.

5. The distributed computer system as recited in claim 1, wherein the network terminal executes Java code on the network terminal.

6. The distributed computer system as recited in claim 1, wherein the same client application is executed on the server computer and the client terminal.

7. The distributed computer system as recited in claim 1, wherein the non-network terminal receives commands from the client application on the server.

8. The distributed computer system as recited in claim 1, including means for passing a client application request to another server to process the request.

9. A method for distributing computing between a server computer and a client terminal which communicate via a network, comprising the steps of:

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

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

(c) responding to the enquiry message at the client terminal with a message comprising 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 computer 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 and services 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.

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

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

12. The method as recited in claim 9, wherein the identification information comprises configuration characteristics of the client terminal.

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

14. The method as recited in claim 9, wherein the same client application is executed on the server computer and the client terminal.

15. The method as recited in claim 9, wherein the non-network terminal receives commands from the client application on the server.

16. The method as recited in claim 9, including the step of passing a 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, comprising:

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

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

(c) a code segment for responding to the enquiry message at the client terminal with a message comprising 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 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 updates the client application and services future requests from the client 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 update of the client application entails a download of the client application to the client terminal.

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

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

22. The computer program as recited in claim 17, wherein the same client application is executed on the server computer and the client terminal.

23. The computer program as recited in claim 17, wherein the non-network terminal receives commands from the client application on the server.

24. The computer program as recited in claim 17, including a code segment for passing a client application request to another server to process the request.

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

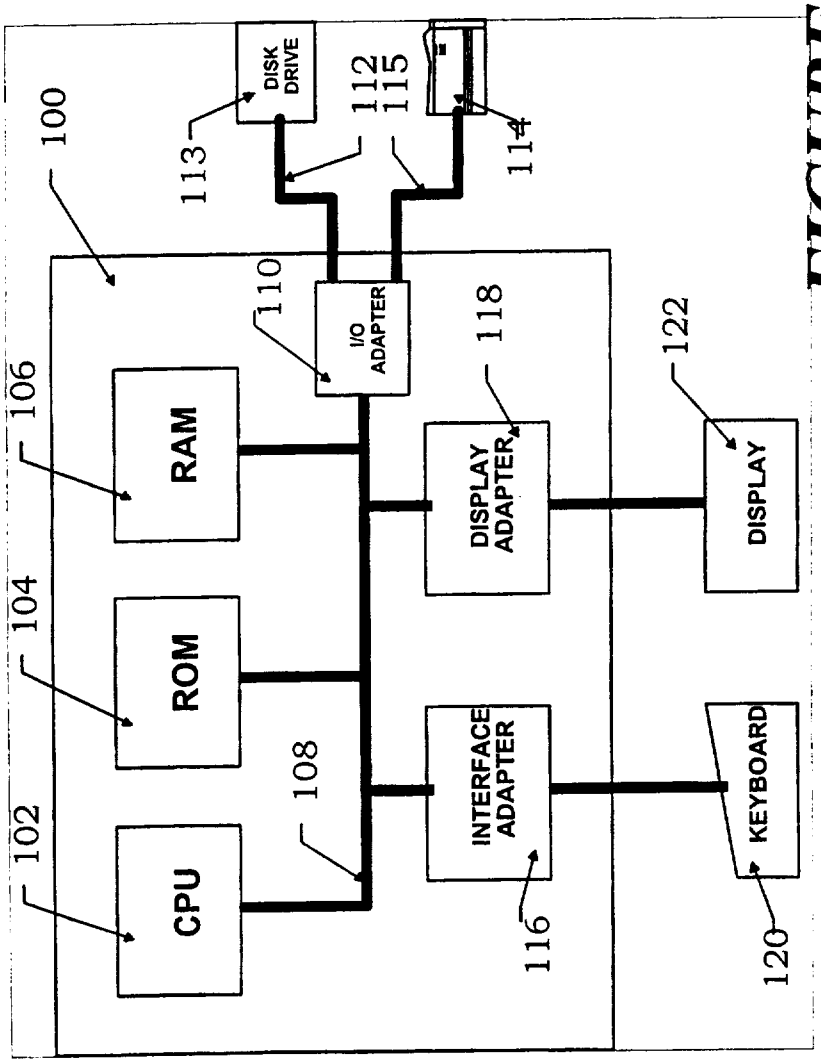

100
102
104
106
108
110
112
113
114
115
116
118
120
122
CPU
ROM
RAM
I/O ADAPTER
DISK DRIVE
DISPLAY ADAPTER
DISPLAY
INTERFACE ADAPTER
KEYBOARD

FIGURE 1

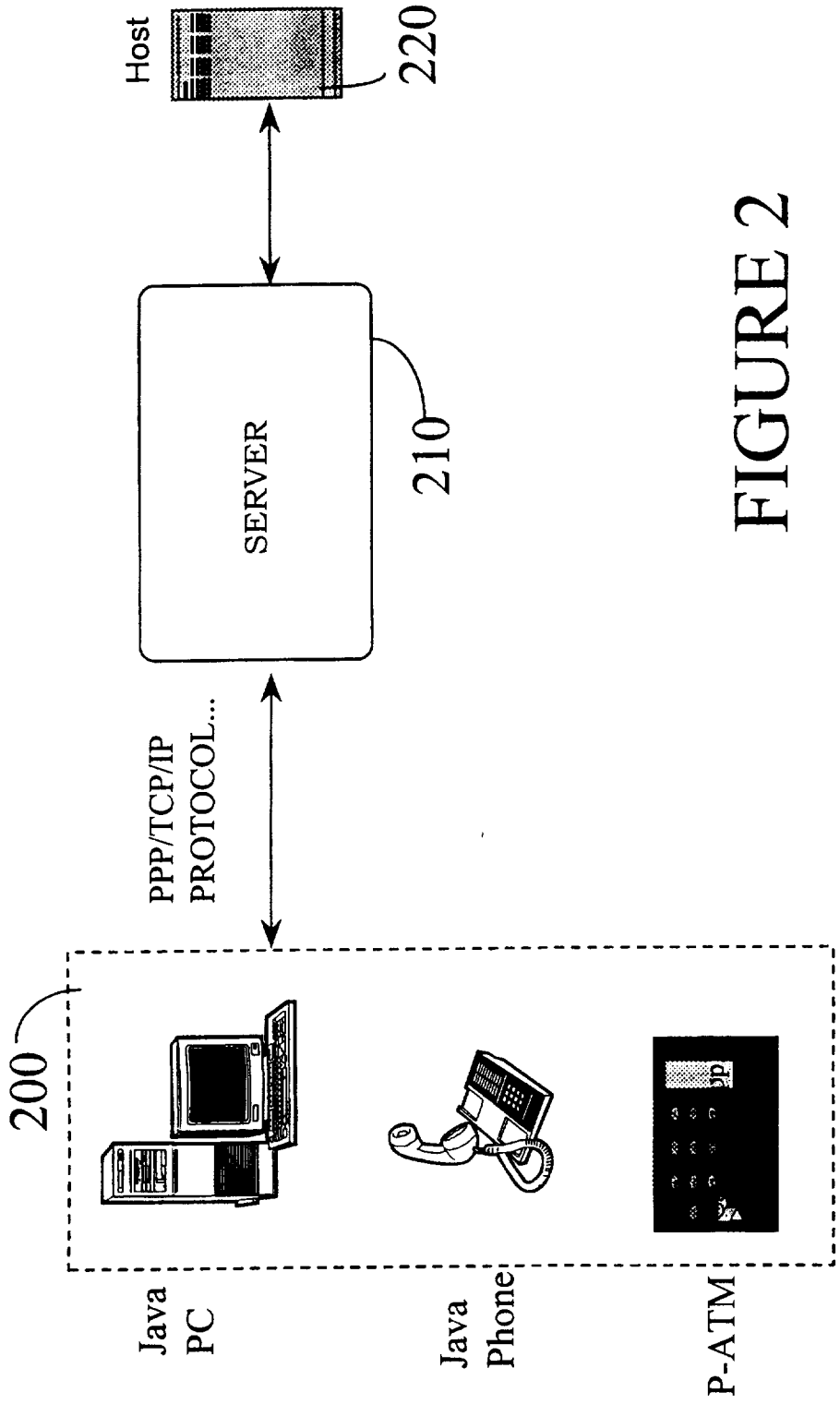

Host
220
SERVER
210
PPP/TCP/IP
PROTOCOL...
200
Java
PC
Java
Phone
P-ATM

FIGURE 2

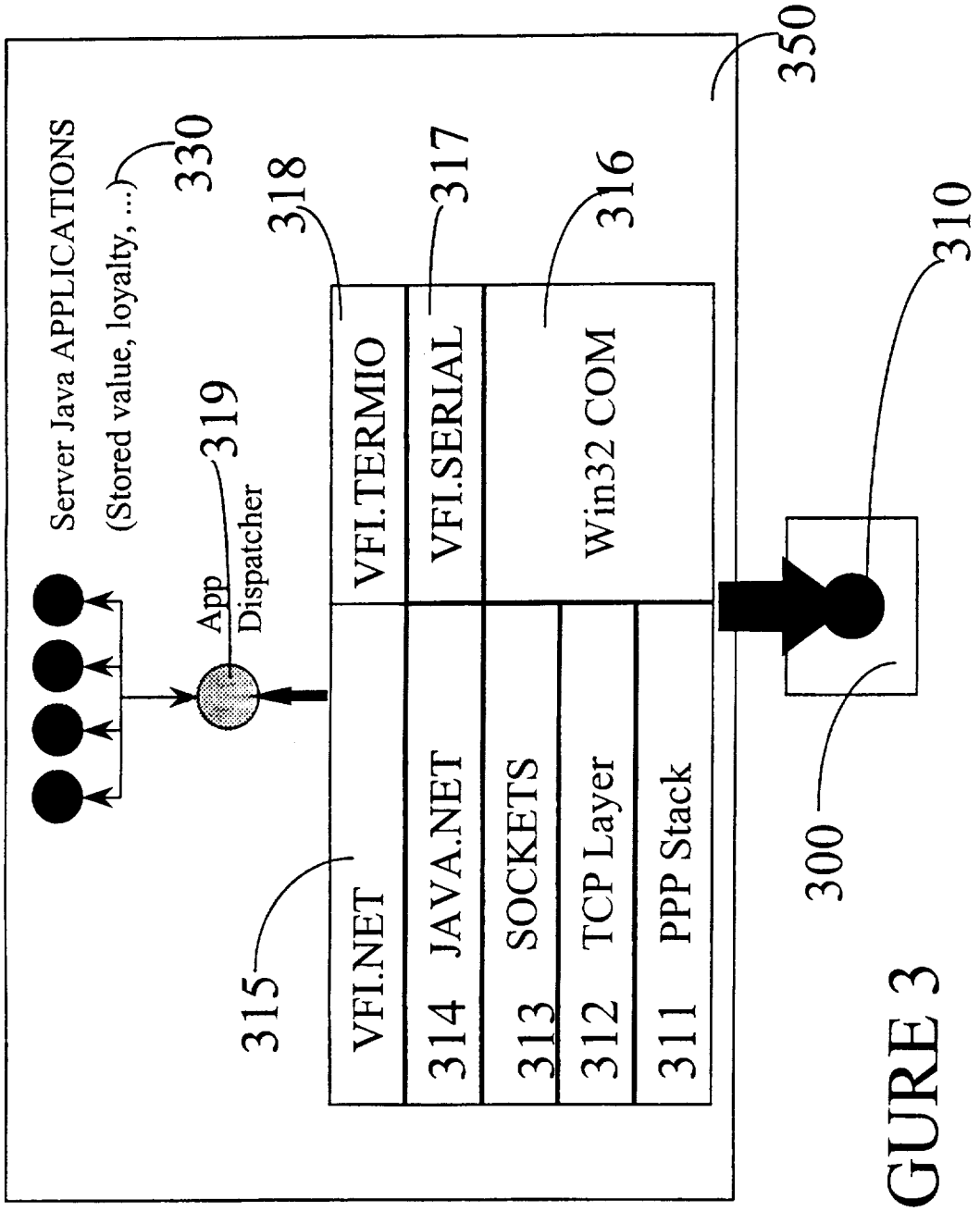

Server Java APPLICATIONS
(Stored value, loyalty, ...)
330
App
Dispatcher
319
318
317
316
350
310
VFI.TERMIO
VFI.SERIAL
Win32 COM
315
VFI.NET
314 JAVA.NET
313 SOCKETS
312 TCP Layer
311 PPP Stack
300

FIGURE 3

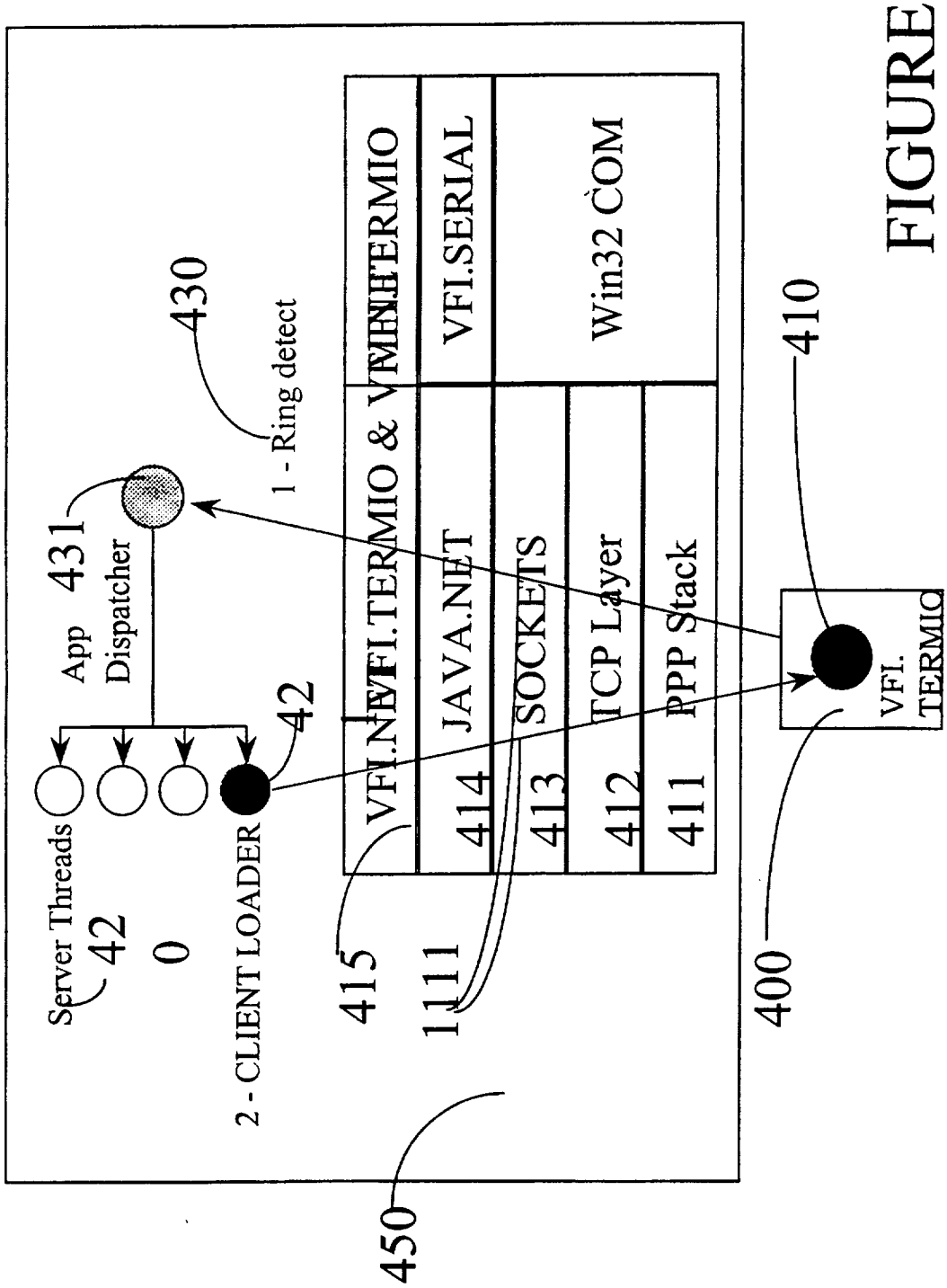

Server Threads
42
0
2 - CLIENT LOADER
App Dispatcher
431
42
1 - Ring detect
430
VFI.NET
VFI.TERMIO & VMIN
VFI.SERIAL
JAVA.NET
SOCKETS
TCP Layer
PPP Stack
Win32 COM
414
413
412
411
415
1111
450
400
410
VFI. TERMIO

FIGURE 4

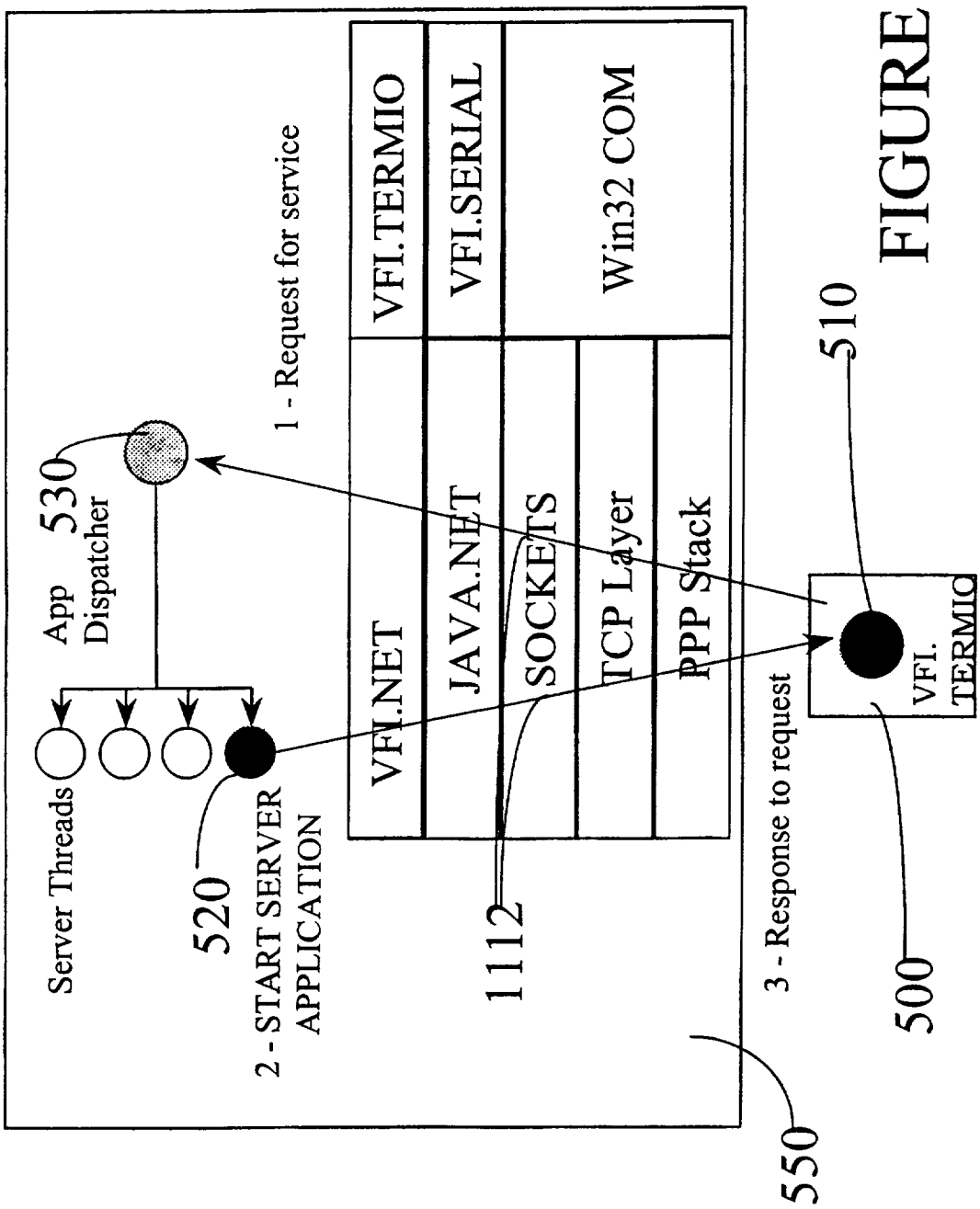

VFI.TERMIO
VFI.SERIAL
Win32 COM
VFI.NET
JAVA.NET
SOCKETS
TCP Layer
PPP Stack
1 - Request for service
App Dispatcher
530
Server Threads
520
2 - START SERVER APPLICATION
1112
3 - Response to request
510
VFI. TERMIO
500
550

FIGURE 5

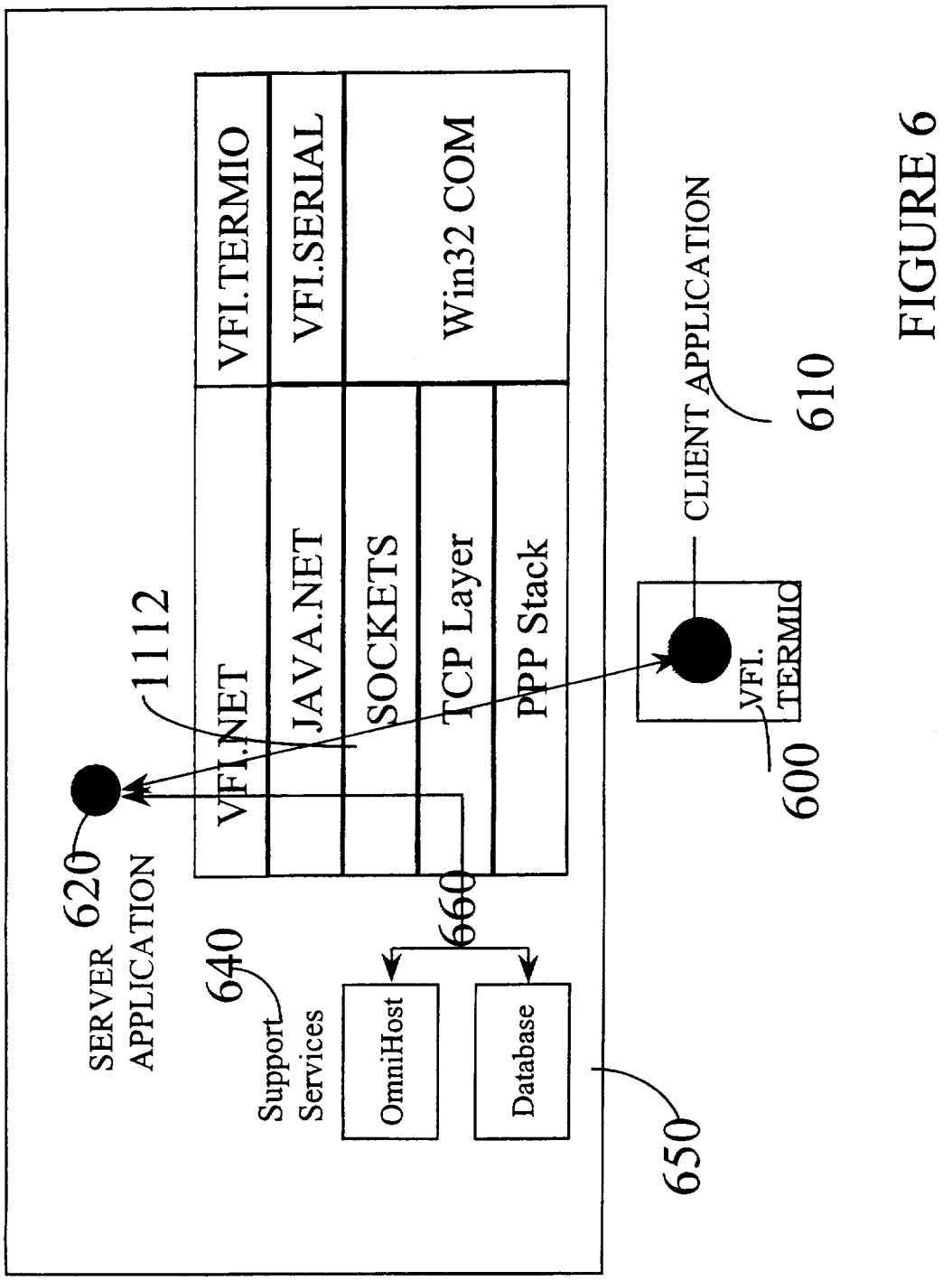

VFI.TERMIO
VFI.SERIAL
Win32 COM
VFI.NET
JAVA.NET
SOCKETS
TCP Layer
PPP Stack
1112
620
SERVER APPLICATION
640
Support Services
OmniHost
Database
660
650
CLIENT APPLICATION
610
VFI. TERMIO
600

FIGURE 6

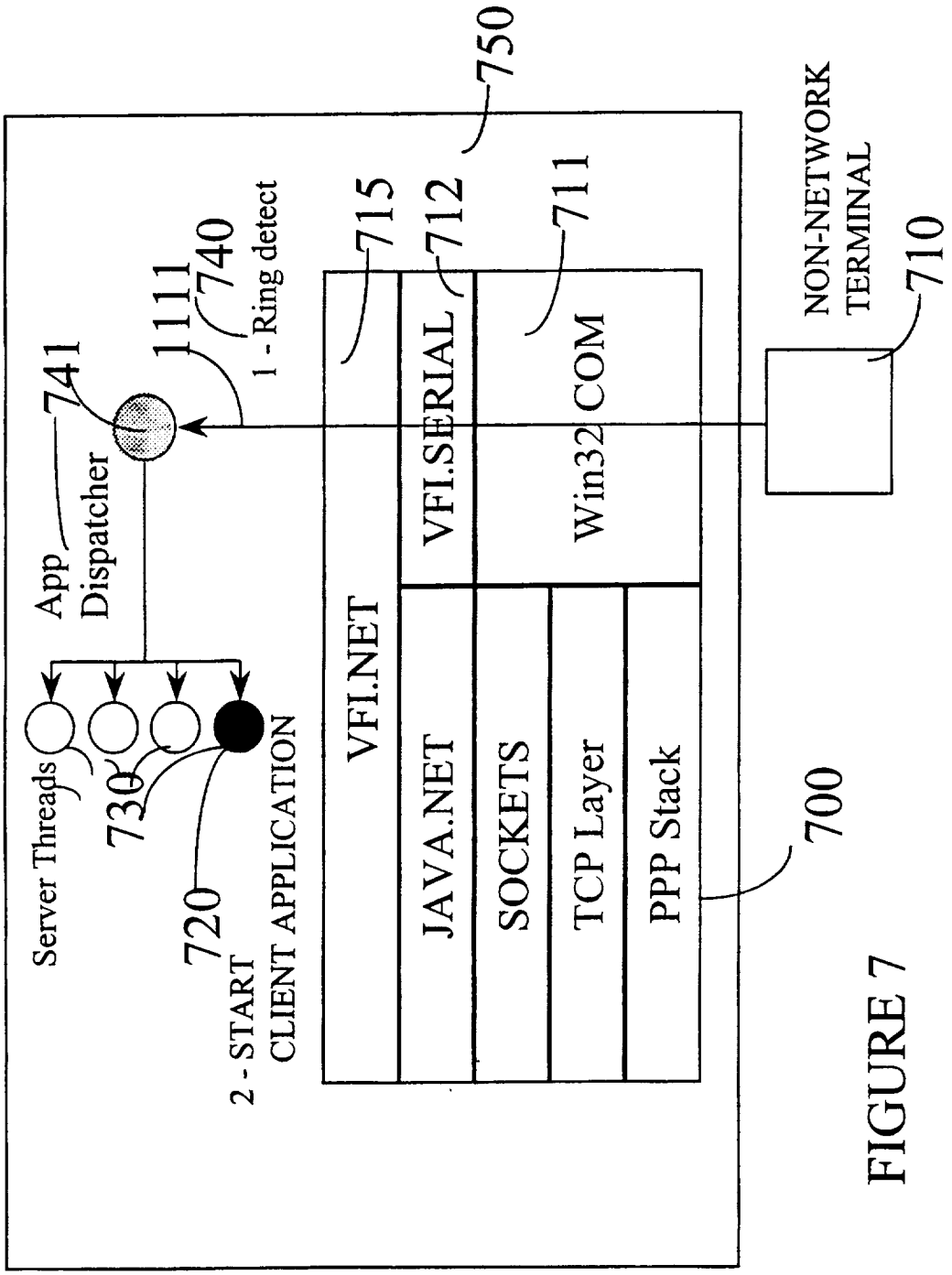

750
NON-NETWORK TERMINAL
710
711
712
715
740
1111
741
App Dispatcher
1 - Ring detect
Server Threads
730
720
2 - START CLIENT APPLICATION
VFI.NET
JAVA.NET
SOCKETS
TCP Layer
PPP Stack
VFI.SERIAL
Win32 COM
700

FIGURE 7

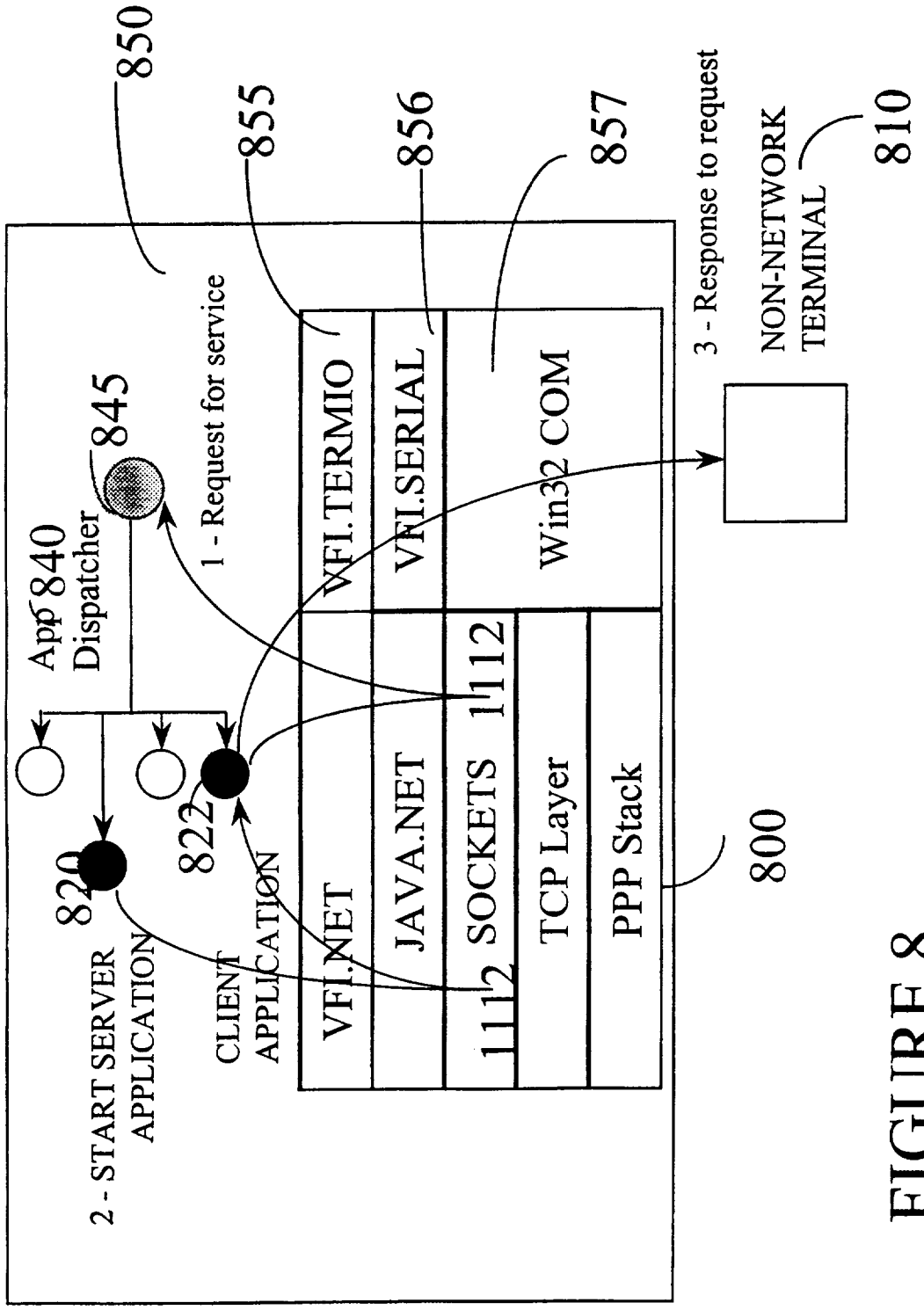

850
855
856
857
845
App 840
Dispatcher
1 - Request for service
VFI.TERMIO
VFI.SERIAL
Win32 COM
3 - Response to request
NON-NETWORK
TERMINAL
810
820
822
2 - START SERVER
APPLICATION
CLIENT
APPLICATION
VFI.NET
JAVA.NET
1112 SOCKETS 1112
TCP Layer
PPP Stack
800

FIGURE 8

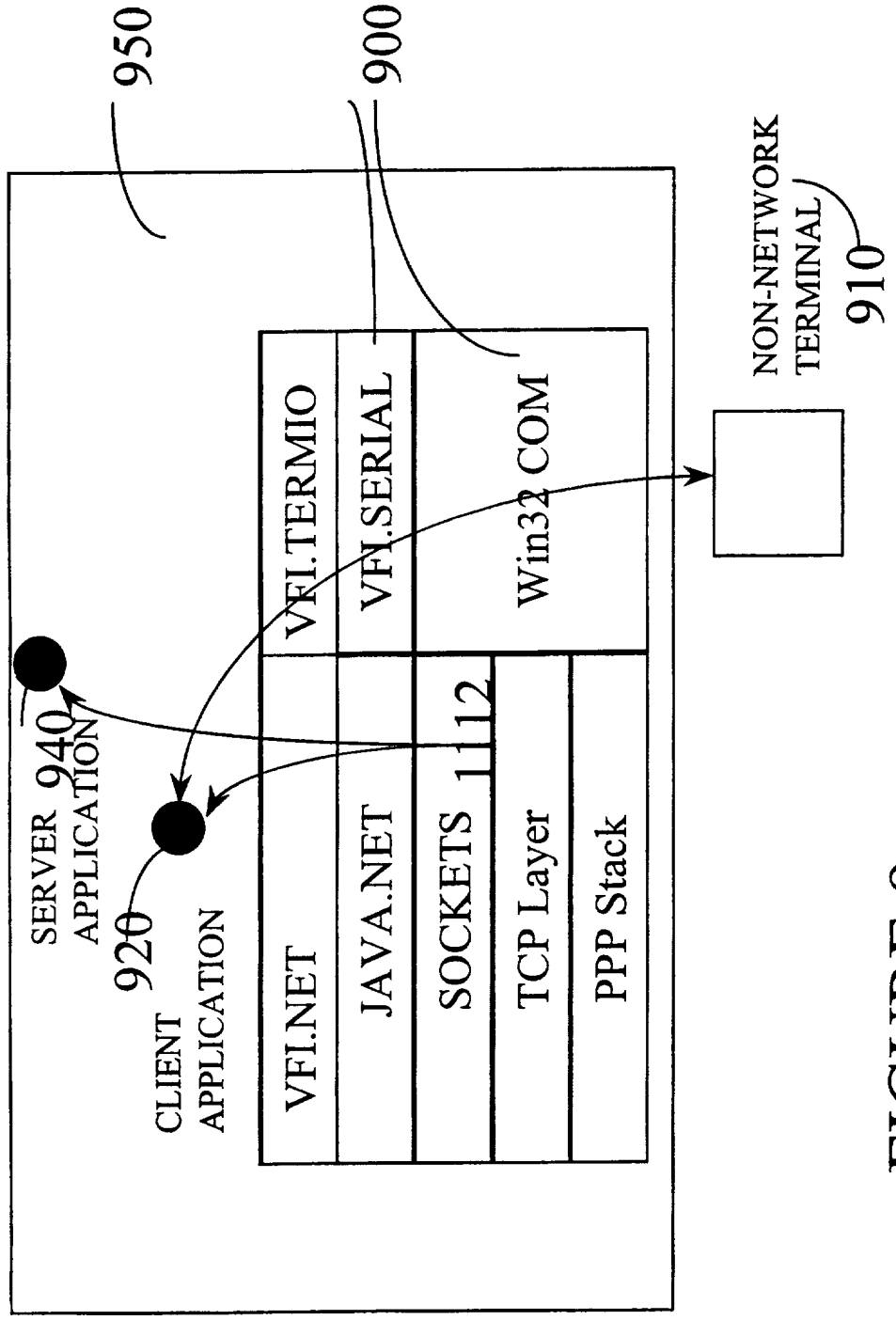

950
900
NON-NETWORK TERMINAL
910
VFI.TERMIO
VFI.SERIAL
Win32 COM
SERVER APPLICATION
940
CLIENT APPLICATION
920
VFI.NET
JAVA.NET
SOCKETS
1112
TCP Layer
PPP Stack

FIGURE 9

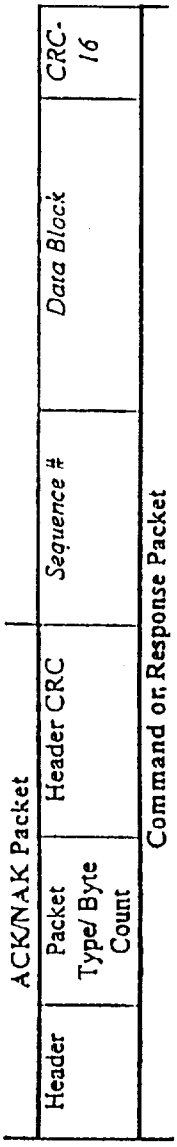

ACK/NAK Packet
Header
Packet Type/ Byte Count
Header CRC
Sequence #
Data Block
CRC-16
Command or Response Packet

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.: Control Packet • ACK • NAK Data (Command or Response) Packet • More Blocks to follow • Last Block • Data Encrypted / Non-encrypted Test Packet • 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

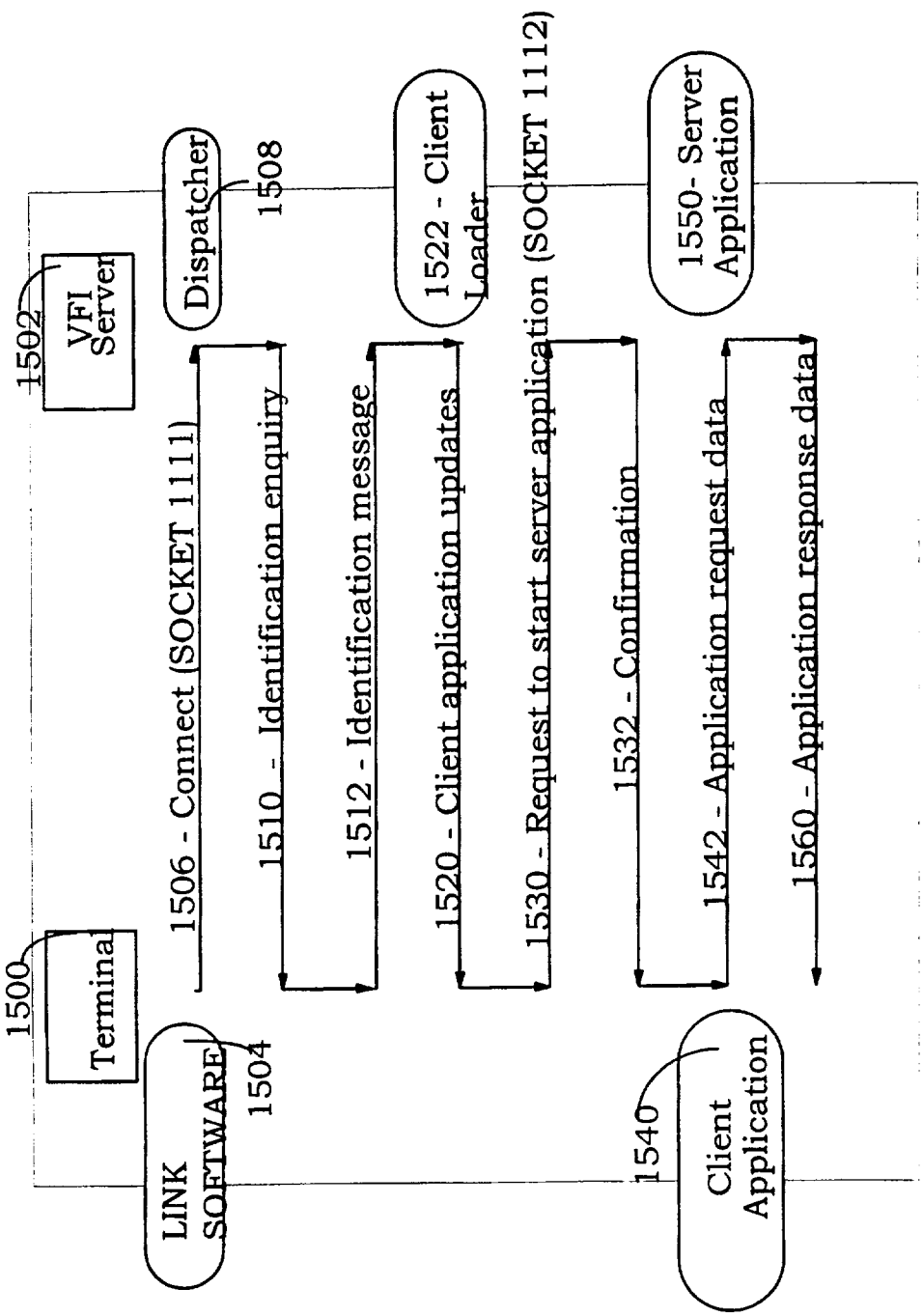

1500
Termina
1502
VFI Server
1504
LINK SOFTWARE
1506 - Connect (SOCKET 1111)
Dispatcher
1508
1510 - Identification enquiry
1512 - Identification message
1520 - Client application updates
1522 - Client Loader
1530 - Request to start server application (SOCKET 1112)
1532 - Confirmation
1540
Client Application
1542 - Application request data
1550- Server Application
1560 - Application response data

FIGURE 15

INTERNATIONAL SEARCH REPORT

Inte. .onal Application No
PCT/US 97/13057

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 H04L29/06

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 H04L

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	EP 0 666 681 A (TRANSACTION TECHNOLOGY INC) 9 August 1995	1-4, 7-12, 15-20, 23,24
Y	see column 8, line 31 - column 9, line 15 see column 29, line 26 - column 30, line 53 see column 32, line 15-49 see column 36, line 10-29 see column 40, line 8-42 --- -/--	5,6,13, 14,21, 22,25

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

[X] Patent family members are listed in annex.

° Special categories of cited documents :

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

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

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

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

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

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

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

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

"&" document member of the same patent family

6

Date of the actual completion of the international search	Date of mailing of the international search report
17 December 1997	14/01/1998

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

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

INTERNATIONAL SEARCH REPORT

Inter onal Application No
PCT/US 97/13057

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
Y A	EP 0 326 699 A (IBM) 9 August 1989 see page 3, line 4-32 see page 7, line 14-16; figure 7 see page 8, line 7-10	6,14,22 1,4,8,9, 12,16, 17,20,24
Y	US 5 506 832 A (ARSHI TAYMOOR ET AL) 9 April 1996 see column 1, line 58 - column 2, line 32 see column 9, line 38-60; figures 1,5,17 see column 48, line 2-30 see column 54, line 55 - column 55, line 32; figure 48 see column 59, line 50 - column 63, line 61; figure 53	5,13,21, 25
P,A	WAYNER P: "Inside the NC" BYTE, vol. 21, no. 11, November 1996, MCGRAW HILL,USA, pages 105-110, XP002050423 see the whole document	1,5,9, 13,17,21

6

INTERNATIONAL SEARCH REPORT

Information on patent family members

Inter onal Application No
PCT/US 97/13057

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0666681 A	09-08-95	US 5195130 A	16-03-93
		AU 6758290 A	13-06-91
		CA 2068336 A,C	10-05-91
		CN 1054164 A,B	28-08-91
		CN 1093475 A	12-10-94
		EP 0499620 A	26-08-92
		JP 7170341 A	04-07-95
		JP 5501645 T	25-03-93
		WO 9107839 A	30-05-91
		US 5485370 A	16-01-96
		US 5572572 A	05-11-96
		US 5321840 A	14-06-94
EP 0326699 A	09-08-89	US 4885789 A	05-12-89
		DE 3855378 D	25-07-96
		DE 3855378 T	23-01-97
		JP 1870558 C	06-09-94
		JP 2007639 A	11-01-90
US 5506832 A	09-04-96	US 5524110 A	04-06-96
		US 5506954 A	09-04-96
		US 5493568 A	20-02-96
		US 5663951 A	02-09-97
		US 5590128 A	31-12-96
		US 5600797 A	04-02-97
		US 5579389 A	26-11-96
		US 5592547 A	07-01-97
		US 5631967 A	20-05-97
		US 5566238 A	15-10-96
		US 5673393 A	30-09-97
		US 5574934 A	12-11-96