PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04L 12/00

A2

(11) International Publication Number: **WO 98/25376**

(43) International Publication Date: 11 June 1998 (11.06.98)

(21) International Application Number: PCT/US97/21745

(22) International Filing Date: 2 December 1997 (02.12.97)

(30) Priority Data:
08/753,827 2 December 1996 (02.12.96) US

(71) Applicant: FIRST DATA CORPORATION [US/US]; Suite 330 AN, 6200 South Quebec Street, Englewood, CO 80111 (US).

(72) Inventors: LAXTON, Gary, R.; 6631 Brunning Glen Court, Charlotte, NC 28215 (US). HURLEY, Thomas, L.; 6635 N. Praying Monk Road, Paradise Valley, AZ 85253 (US).

(74) Agents: LAURIE, Ronald, S. et al.; McCutchen, Doyle, Brown & Enersen, Three Embarcadero Center, San Francisco, CA 94111 (US).

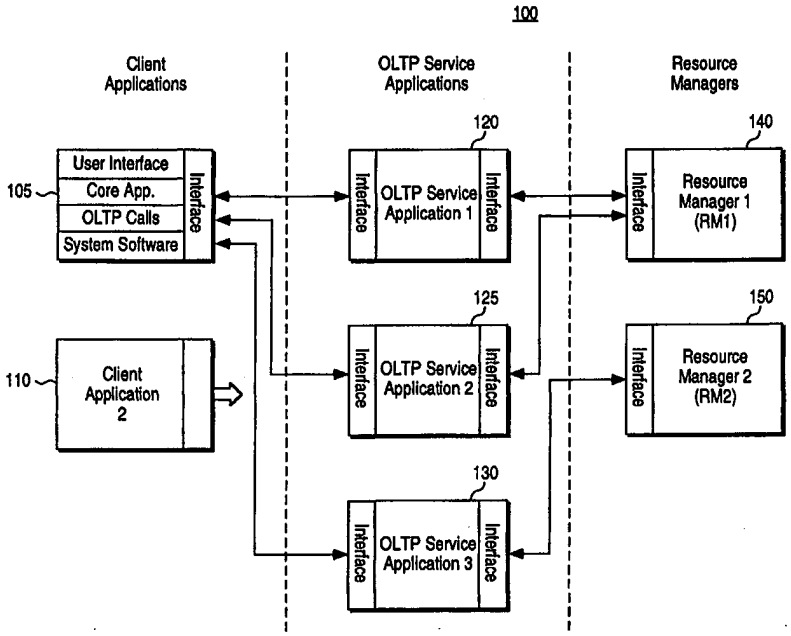
(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (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
Without international search report and to be republished upon receipt of that report.

(54) Title: METHOD AND APPARATUS FOR IMPROVED TRANSACTION PROCESSING IN A DISTRIBUTED COMPUTING ENVIRONMENT

(57) Abstract

Enhancements to on–line transaction processing (OLTP) for distributed computing environments. Improved mechanisms for implementing OLTP systems which are hardware and operating system independent and which provide flexibility in architectural design. Remote clients may transparently communicate with OLTP software through a versatile remote procedure call implementation. Network bandwidth usage is minimized through a dispatcher function which groups client transactions at a dispatcher service. Message passing is handled through flexible message object technology which is data type and operating system independent. Finally, a WORLD WIDE WEB browser interface is created for connecting various user clients to appropriately configured OLTP systems.



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		

METHOD AND APPARATUS FOR IMPROVED TRANSACTION PROCESSING IN A DISTRIBUTED COMPUTING ENVIRONMENT

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to on-line transaction processing (OLTP) in distributed computing environments. More particularly, the present invention relates to improvements and enhancements to OLTP techniques for transaction handling in a heterogeneous computing environment.

2. Background of the Invention

Business-critical software applications are generally transaction oriented and have exacting performance, integrity and administrative requirements. These requirements dictate an architected approach to application development, deployment and operation. In recent years, a component software model has emerged from the software industry to provide an architectural solution to business-critical processing in distributed computing environments which perform on-line transaction processing (OLTP). Newer OLTP architectures carry forward the legacy mainframe attributes of availability, integrity, performance, scalability and manageability, but in true distributed applications. Implementations of a component-based OLTP application design creates a "virtual mainframe" from heterogeneous computing resources thus providing a managed coupling of resources at the distributed application level.

In a distributed computing environment, each application must be able to communicate and exchange information with other applications or machines in the environment. If all of the machines are based on the same hardware platform, use the same operating system, and are interconnected using a single network architecture and communication protocol, connection and communication between applications and/or machines is straight forward. However, this ideal is seldom achieved. There are many different, often mutually incompatible, computer architectures, hardware platforms, operating systems, and application languages utilized for various functions in a distributed OLTP system. This heterogeneity presents an obstacle to the connectivity and interoperability of the systems.

Many organizations now recognize that the component software model is the logical evolution of early distributed computing efforts. Mainframe and centralized mid-range application systems are now augmented by a large base of powerful desktop and server systems. These distributed systems are loosely coupled at the network level by standard network transports, and form a network computing resource infrastructure. Initially, this infrastructure served to migrate "front-office" applications from the centralized systems, largely the document processing and email communication applications easily implemented within desktop processors and file servers. Next, two-tier client-server database applications that also utilized the distributed environment were deployed at the departmental level. These applications supported finer granularity at the data level, moving from interactive file sharing to concurrent data element access. These first client-server applications

provided proof-of-concept for true distributed application processing, but remained *ad hoc* in nature and of limited scale and manageability. Effective distributed processing for larger, more business-critical applications requires a shift from the previous client-database only approach.

The component software model extends client-server computing by supporting application partitioning to more effectively develop and deploy application core logic and manage its reliable execution in a network environment. The component software model's three-tier architecture promotes a clean separation of functionality: presentation/user interaction, core application logic and data management. This separation confers benefits to the application developer and supports optimized execution of the overall distributed application. The development benefit of the decoupling presentation and core application logic is twofold and grows proportionally with the number of clients participating in the application. First, component development allows application designers to focus on encapsulating core application functions in their workflow and interaction. Second, clean presentation/application logic functional separation supports productive life cycle management of the application code. Frequent distribution of updated desktop applications to hundreds of clients is a serious maintenance problem for many client-server applications.

With the component software model, presentation elements such as data input and display means are decoupled from the internals of the application operating logic. Changes to application logic encapsulated in the various services often will not require a change in the client side

application. In recent years, several vendors have developed component software OLTP architectures including the Tuxedo® system from Novell and the Encina® system from Transarc Corporation. While these systems overcome a number of disadvantages of prior distributed computing environments they leave unresolved several issues such as the growing trend toward ever "thinner" client applications, and create one new disadvantage which is a potential dependence upon a particular vendor of OLTP technologies. In addition, the existing OLTP platforms leave unresolved certain problems generated by the interaction of differing hardware and operating system platforms and the need to minimize the usage of a limited resource which is network bandwidth. Thus, there is room for improvement in the development of business-critical distributed computing environment platforms.

SUMMARY OF THE INVENTION

The drawbacks of the existing on-line transaction processing (OLTP) systems are overcome by the methods and apparatus of the present invention. In a first aspect of the present invention, a mechanism is provided for the seamless separation of client software from a subsequent layer of client or server software in the OLTP system. The remote user client and the system to which it must communicate are each provided with a layer of interface logic whereby the remote client software has its calls to the application service software trapped by a remote procedure call layer. The remote procedure call layer is implemented to create a connection to the core service system upon which the application level software is operating for transferring the appropriate calls. The invoked service routines are then carried out by the application in a manner which is transparent to the fact that the initiating client is remote.

In another aspect of the present invention, network communications between client applications and OLTP service applications is minimized by introducing an additional service application which translates single client instructions into multiple server transactions and takes responsibility for transaction management in calling the other server processes thus minimizing the interaction between a client and the various server services as well as minimizing the size of the client application code. This dispatcher service is table driven and uses such tables to map the single client procedural call to the multiple application transaction services. This also provides for an efficient mechanism of

changing the sequence and operations carried out at the server without having to change client application software.

In yet another aspect of the present invention, certain incompatibilities between differing hardware and operating system platforms are handled in a transparent manner which improves the flexibility of message passing in the OLTP system. Rather than using the tightly coupled message passing schemes promoted by OLTP system vendors, a dynamic messaging scheme is provided which creates a message object for sending messages between the client and server applications which are data-type independent as well as content order independent. This aspect of the invention allows applications running on numerous different operating systems on different hardware platforms to interact in a seamless manner while providing application developers the opportunity to be flexible in defining their message passing requirements.

In a final aspect of the present invention, the effort to create a "thin" client is taken to the extreme by introducing an OLTP service which provides a world wide web interface to the OLTP system. This is known as an HTML web API which allows remote users access to the OLTP system through nothing more than an off-the-shelf web browser. The above-described message object is the key to the HTML transaction API. By invoking a CGI program or script in response to the HTML request, the web server is capable of operating as the OLTP client to the rest of the OLTP system while providing web server functionality to the web browser client. The CGI interface interprets OLTP requests from the HTML pages which are parsed with a specified nomenclature to create the OLTP message objects which are then provided to the

OLTP system for transaction processing. Similarly, the CGI interface is used to generate the HTML response to the web browser. One technique implemented is to embed the HTML extensions in HTML comment fields, marked by special delimiters. The result is to allow an end user access to the OLTP system with nothing but a commercial, off-the-shelf web browser.

BRIEF DESCRIPTION OF THE DRAWINGS

The objects, features and advantages of the present invention will be apparent from the following detailed description in which:

Figure 1 illustrates a logical diagram of an OLTP system architecture.

Figure 2 illustrates a computer system architecture in which OLTP client software is implemented in the same computer system as the OLTP service software.

Figure 3 illustrates one embodiment of an aspect of the present invention wherein the client application for the OLTP system is made transparently remote from the OLTP service portion of the OLTP software.

Figure 4 illustrates an OLTP system implementing a dispatcher service functioning in accordance with another aspect of the present invention.

Figure 5 demonstrates an example message exchange between a client and a specified service using a message object generated in accordance with the teachings of the present invention.

Figure 6 illustrates the interaction of a workstation operating a web browser with a web server over the Internet's World Wide Web.

Figures 7(a) and (b) illustrate alternative embodiments of an HTML server configured to interact with the OLTP system of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

Methods and apparatus are disclosed for an on-line transaction processing (OLTP) system implemented in a distributed computing environment. Throughout this detailed description, numerous details are set forth such as specific data types and various communication protocols in order to provide a thorough understanding of the present invention. To one skilled in the art, however, it will be understood that the present invention may be practiced without such specific details. In other instances, well-known structures and devices have not been shown in detail in order not to obscure the present invention. In particular, much of the functionality to be described herein is carried out by software running on various computer related components in a transaction processing system. With current software development techniques and component and object-oriented approaches to software architecture, the description of component software products does not generally require the presentation of fully detailed source code listings. The explanations of logical functions to be carried out by various software components and their interaction with other portions of the system are sufficient to allow software designers of ordinary skill to design systems meeting the functional requirements. Accordingly, the present invention will be described primarily in terms of the various functionality to be implemented by various components of the transaction processing system. Those of ordinary skill in the art, once given the following descriptions of the various functions to be carried out by the present invention will be able to

implement the necessary programming logic in various programming languages or through other techniques without undue experimentation.

A portion of the disclosure of this patent document contains material which may be subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent disclosure as it appears in the Patent and Trademark Office patent files or records, but otherwise reserves all copyrights whatsoever.

Referring now to Figure 1, there is illustrated an exemplary OLTP system 100. In an OLTP distributed computing environment client application components provide user interface routines and transaction request services. Application services encapsulate core processing logic, providing services such as transactional data access and updates, etc. The OLTP services fulfill transactional requests by interacting with resource managers, which may include database systems provided by companies such as Oracle and Sybase, etc. Thus, in Figure 1 there is illustrated client applications 105 and 110 which may operate on workstations or other intelligent desktops such as Unix, DOS/Windows, MAC, or OS/2 systems and may also include point-of-sale terminals, automated teller machines and other devices optimized for specific data gathering and presentation functions.

The primary client function is to gather information for subsequent processing and to display information either input or returned as the result of transaction processing. Transaction functions are accessed as named services advertised to the client application. The client application

generates and packages request messages which are directed to a specific application service. One aspect of the present invention that will be discussed more fully below is a robust message passing mechanism which allows for otherwise incompatible operating systems and hardware platforms to communicate in a transparent manner. In large-scale applications, many clients may concurrently request transactions from the OLTP service components for processing by the respective encapsulated services.

The exemplary OLTP system illustrated in Figure 1 shows three OLTP service applications 120, 125 and 130 each respectively available to perform services advertised to the clients 105 and 110. Application service components encapsulate the core logic of their respective functions in the form of named service routines. Service routines are typically dedicated to a single type of task. Service routines encapsulate data access, for example, by including embedded SQL or database stored procedure logic. Clients may request data update or retrieval in the form of a service request to the particular server or service application arbitrates resource manager interaction on behalf of the client. A server component may advertise multiple individual service routines, each of which may be specifically invoked by an authorized client application. The distributed computing environment OLTP system typically comprises one or more application service components, each grouping the service routines for a related group of actions or tasks constituting an identifiable transactional function. Any number of concurrent, authorized clients may issue requests for service from the server application tier. A number of existing OLTP managers such as Tuxedo® and Encina® mentioned above provide

lower level mechanisms for service tracking, priorities, load balancing and available resource checking for their costing and scheduling decisions.

In general, application services are dedicated to individual resource managers, although multiple application services may access the same resource manager. Thus, there is illustrated in the exemplary embodiment of an OLTP system in Figure 1 a resource manager 140 which is configured to communicate with application services 120 and 125, respectively. The second resource manager 150 is shown as a resource manager for application service 130. Data are stored in data "resource manager" such as resource managers 140 and 150 which may be relational databases, indexed file systems, store and forward queue systems and other such resources. The application service level software such as servers 120, 125 and 130 service routines as a "clients" to the various resource managers or data servers issuing the SQL, enqueue/dequeue requests or other appropriate data access protocols for the specific resource manager.

OLTP systems in implemented embodiments are generally far more complex than the exemplary embodiment illustrated in Figure 1. In fully operational systems, there may be thousands of client applications communicating with many application services and numerous resource managers. In addition, the calls to one application service may implicate calls to other application services which may implicate interactions with multiple resource managers servicing a single set of procedure calls from a client application. In addition, multiple instances of service routines may be required with the number varying as loads change and the

system requirements dynamically change. It is the nature of having to handle multiple resource managers which has led to the most dramatic changes in OLTP services. Particularly, the ability to have a two-phase commit sequence. In a two-phase commit architecture, it is possible to rollback the state of the system to a synchronization point if a commit instruction to any of the resource managers fail. An industry standard interface, known as the XA interface, supports a two-phase commit which allows such complex OLTP systems. Much of the teachings of the present invention are directed toward XA compliant OLTP architectures.

Remote Procedure Call Support For Remote Clients

One common drawback to existing OLTP systems are the mechanisms for providing a client interface with the core service routines as embodied on a single server machine. While the single machine may be able to support multiple server processes, it is desirable to provide a simple mechanism for a client application to be on a remote machine from the machine executing the service routines. Figure 2 illustrates a prior art configuration of an OLTP application service in which a single Machine A 200 supports an OLTP server core for calling multiple service function instances or procedures 220 which in turn communicate with database resource manager interfaces 230 to direct resource requests to a resource manager 240. In the early prior art system of Figure 2, the OLTP client application resides on the Machine A as OLTP client 205. The OLTP client 205 in the prior art is tightly coupled to Machine A for invoking advertised services 220 for functions to be carried out at resource manager 240.

In this system, the service applications are required to be running on the same Machine A as the OLTP client application 205.

Figure 3 illustrates an architectural level diagram of a mechanism of the present invention for removing the OLTP client application core from Machine A 200 to a remote Machine B 300. In this case, the OLTP client 300 makes the same OLTP procedure calls as the client 205 so that they may utilize the same code that was compiled on Machine A 200, but compiled to run independently on Machine B 300. However, in the utilization of a remote client, there is introduced a Client RPC interface 305 on Machine B 300 and a Client RPC interface 310 on Machine A 200. The Client RPC routine 305 traps calls to the OLTP service core 210 that are made by the OLTP client application 300. The Client RPC routine 305 creates a socket connection to Machine A 200 over the communication network linking Machines A and B for forwarding the OLTP service requests to the Client RPC code 310 on Machine A. This provides an interface to a portion of the client application 320 remaining on Machine A 200 which is responsible for presenting OLTP service calls to the OLTP service core 210.

A Dispatcher Service For Minimizing Client Messaging

In a structured software architectures supporting a distributed computing environment, it is desirable to make the client applications as "thin" as possible. That is, it is better to have as much processing logic as possible in server-based programs which occur in much fewer instances than in client-based applications which are much more cumbersome and

voluminous in nature when it comes to upgrading and refinements. Another object of distributed computing environments is to implement mechanisms which reduce the amount of inter-application communication because network resources are limited. Client applications should be prevented, whenever possible, from monopolizing such resources. There is thus disclosed in Figure 4 an exemplary OLTP system which provides a refinement to existing OLTP systems in order to minimize the number of transactions between the client and the server levels of the OLTP system and which provides additional advantages. In the exemplary embodiment OLTP system 400, two server routines or service applications are illustrated: the Procedure One Service 410 and the Procedure Two Service 420. The client application 405 may present transaction options to a OLTP that require invocation of multiple server-based application services such as services 410 and 420. Not illustrated in Figure 4 are the resource managers to which the respective service procedures 410 and 420 would direct their respective data requests.

The illustrated OLTP system 400 of Figure 4 also shows a facet of OLTP systems wherein multiple instances of various service procedures may be executing. In this example, application service 410 is shown operating with two instances and application service 420 is shown operating with three instances. This allows multiple clients to call a service application and reduces the odds that the service will be busy.

When a service application begins executing, it broadcasts its availability and associated information such as resource manager information and available service procedures

to a bulletin board service 430. The bulletin board service 430 creates and maintains a table 435 that identifies all operating services in the OLTP system and whether or not the particular service is busy. Thus, table 435 illustrates that instance one of Procedure One is busy, denoted by a 'B' in the table, while instance two of Procedure One is ready denoted by an 'R'. Similarly, in the example illustrated in Figure 4 all three instances of Procedure Two are shown to be ready to receive calls.

An enhancement to OLTP systems illustrated in the OLTP system 400 is the introduction of a Dispatcher service 440. The function of the Dispatcher service 440 is to group together sequences of service invocations thus reducing the amount of processing necessary by the client application 405 and reducing the number of messages transmitted from and to the client application 405. The Dispatcher service 440 maintains a table 445 which associates the client call with the actual service procedure calls to be executed. Thus, if a client application transaction actually implicates two service procedures, it need make only a single transaction call to a designated service name and the Dispatcher service 440 will look up in table 445 the actual service procedures to invoke to initiate the appropriate service application. In this context, the Dispatcher service 440 coordinates with the bulletin board service 430 to select a managed server process to make sure the procedure calls wind up in the correct service procedure queue. The collective reply messages from the multiple service procedures may then be received by the Dispatcher service 400 and lumped into a single reply message back to the client application 405, thus minimizing the number

of messages passed between the client application level and server application level of the OLTP system.

By implementing a dispatcher service that is table driven, it is possible to dynamically map requests to different managed servers without requiring updates to client applications software. Thus, if an improved Procedure One service is provided elsewhere in the OLTP system, the table 445 can be updated to identify an alternative application server which will have presumably broadcast to the bulletin board service 430 that it is available.

Hardware And Operating System Independent Transparent Message Object Passing Mechanism

One impediment to a truly flexible, scaleable OLTP distributed computing environment is dealing with the fact that different computer platforms running different operating systems often define various data types differently. For example, a Unix based machine may define an integer to be a four byte piece of data with the order of data occurring from high byte to low byte. In contrast, a PC-based machine may define an integer data type to be two bytes long, ordered from low byte to high byte. The present invention provides a message passing scheme which is transparent to hardware and operating system platforms and does not require a tight coupling between client and server applications.

A conventional OLTP message passing call might look like something:

```
OLTPcall ("ServiceName", ibuff, ibl,
    obuff, obl, flags)
```

Such a call exemplifies prior message passing techniques used by many OLTP systems where the data sent to a particular service application, in this case "ServiceName", is a raw data buffer, ibuff, which must be correctly interpreted and decoded by the receiving server. In the case where the server operates in a different operating system or different hardware platform, it must be programmed to be aware of the order and data types for the information presented. Similarly the return buffer, obuff, suffers the same constraints because it must be properly decoded by the client application machine to ensure proper data passing. Other OLTP systems provides specific routines to code the message buffers in such a way as to be decoded automatically by the OLTP core software when the buffer is sent to another hardware platform. However, these other buffering schemes are still very tightly coupled between the client and the server code in that they dictate very precise data ordering rules.

In accordance with an aspect of the present invention, a dynamic message passing scheme is disclosed in which message buffers may be readily interpreted because the data type information is included in the message buffer and is transmitted with the message to the remote application code. Using an object oriented programming language such as C++, it is possible to create a message object containing all of the relevant information. Figure 5 illustrates an example of a screen update application on a work station for a financial transaction. At the Client Machine A 500, the client application provides an interface by which a user may enter

the user's name and account number on the display screen at 505. Then, at step 510 the user name and account number are taken from the screen and placed into a message object 575 and sent to an application service, in this example titled the AccountInfo Service which runs on the account information server Machine B 520. The application service at step 530 retrieves the account number and name from the provided message object and generates a transaction request to the resource manager 540 to retrieve the account balance. Then, at step 550 the AccountInfo service takes the data from the resource manager and adds that to the message object 575 which is then returned to the client application at Machine A 500. At step 560, the account balance is retrieved from the message object 575 which has been received and is presented back to the user interface for presentation to the user.

For the example of Figure 5, the message object is created by a C++ object, in this case called OLTPMsg, which in essence provides the following C++ calls:

```
OLTPMsg.Put (DE_USERNAME, Gary Laxton)

OLTPMsg.Put (DE_ACCOUNT_NUMBER, 123456)
```

The application stores the provided data in the message object, is graphically illustrated by object 575 of Figure 5. The buffer is filled with the identification of the data variable, its data type, its length and the particular data. Thus, regardless of the kind of machine running the client application at Machine A or the server application at Machine B, the data type for the variables conveyed with the data itself is provided in a manner that each operating system can resolve upon reading. After the input data has been put

into the message object 575, the service procedure AccountInfo is engaged by another call to OLTPMsg such as:

```
OLTPMsg.SendMsg (AccountInfo)
```

which transmits the message object 575 to the OLTP service named in the SendMsg routine.

This method of generating a message object allows the client to change the message by adding different fields etc. without having to recompile the service application code. It also allows the message object to be filled in a matter that is order independent. When the AccountInfo service 520 receives the message object, it may retrieve the values placed therein by calls to OLTPMsg.Get ('name of variable'). The server will then do an OLTPMsg.Put to place the retrieved data into the field DE_ACCOUNT_BALANCE and return that to the application Machine A. Similarly, client application will retrieve the value returned with a call such as:

```
OLTPMsg.Get (DE_ACCOUNT_BALANCE,
return_val)
```

which will return the account balance to the user interface for presentation.

Thus, by using a C++ like message object for data passing between machines, data type transparency and operating system/machine transparency are provided. Further, the order and sequence in which data are placed in the message object are inconsequential to either the operation of the client application or the server application.

World Wide Web Client API For Use With OLTP System

In recent years there has been a dramatic shift in the way in which computers are used and data is accessed. Network technologies have expanded the interconnectivity of the computer world providing almost immediate access to information stored anywhere in the world to be displayed on a local desktop almost anywhere else in the world. The sudden super vitalization of the Internet and the popular mechanism for utilizing it known as the World Wide Web have completely changed the notion of client server technologies. Today, the notion of an extremely thin client is exemplified by personal computers running World Wide Web browsers such as Netscape Navigator and Internet Explorer by Microsoft. Applications may be written entirely for operation on a web server for presentation and operation on a user's computer without ever having to change or install new software at the user client.

Figure 6 illustrates an exemplary World Wide Web computing environment 600. In operation, a user may use a personal computer or workstation 610 which includes its standard operating system software 615 over which runs web browser software 620. The web browser 620 provides an interface for a user to specify by name or address a location on the World Wide Web which contains a server to which the user desires to connect. In what is becoming common terminology, the user of user station 610 provides a universal resource locator (URL) which the web browser 620 uses to initiate the connection to the desired server 630. The web browser 620 communicates with the desired server over the Internet or Intranet 620 or other implemented computer network utilizing a standard protocol referred to as the hypertext

transport protocol (HTTP). The machine with which the web browser communicates is generally a server 630 running web server software 640 which is responsible for connecting the server to the Internet 625 for communication with remote web browsers anywhere in the world. The web server software 640 provides rendering instructions with presentation materials defined in accordance with the hypertext markup language (HTML) to the web browser 620 over the Internet 625 using HTTP. These are frequently referred to as a HTML pages.

Use of the World Wide Web today has become increasingly interactive. A user may provide information through a user interface on the web browser which can be packaged and transported over the Internet to the web server 640. This may effect the selection or presentation of the HTML pages 650 which return information back to the web browser 620. HTML has been created to be a fairly simple programming language which allows people who are less than fully trained software engineers to create dynamic programs for presentation to end users on their web browsers.

There is another mechanism that makes World Wide Web programming extremely dynamic, and that is the Common Gateway Interface (CGI) which is a programming interface provided by web servers to allow calls to external procedures outside of the web server software. Thus, rather than strictly retrieving and returning HTML pages 650, the web server 640 may interact with CGI programs 660 which provide a more dynamic programming environment. CGI programming is substantially more complex than HTML programming and requires a greater proficiency and training in software engineering. A CGI programmer will generate results to be presented on a web

browser by programming the various CGI logic to be carried out while building a resulting HTML page to provide to the web server 640 for return to the web browser 620. One disadvantage to CGI programming, obviously, is that it sacrifices the ability to let more designed-oriented individuals who are capable of HTML programming from creating products which require CGI techniques.

In another aspect of the present invention, a mechanism is provided which allows for a more robust interaction between HTML pages designed by an HTML designer and CGI programs developed by CGI programmers. In addition, this aspect of the present invention allows a web server to operate as a "client" to the above-described OLTP systems while implementing the extremely thin client approach provided by the World Wide Web.

The message object technology described in the previous section is a key ingredient to the HTML extensions provided by the present invention.

As described, there is a special interface, CGI, supported by web servers that allows a developer to satisfy web browser requests programatically. To accomplish this the web server sends the requests to a user specified program (CGI script) which can format data into HTML form to be returned to the browser for display on the user screen. To make a web server capable of interacting with the OLTP systems of the present invention, the CGI interface is used to receive requests from the browser and format them in an appropriate message objects to send to the OLTP system. Figures 7(a) and 7(b) show two alternate embodiments for a web server implemented in accordance with this aspect of the present

invention. In Figure 7(a) the web server 700 includes web server application software 710 that communicates with HTML pages 720 and CGI programs 730. The CGI programs are written in such a manner that they are able to take the data provided by the client application web browser, package them into a message object such as that described above, and direct them to a desired OLTP service process or to the Dispatcher service described above with respect to Figure 4. Thus, the web server 710 itself becomes a "client" to the OLTP system. It can be seen from this architecture of Figure 7(a) that nothing is required to be changed at the end-user client system in order to operate with the OLTP system.

Figure 7(b) presents an alternate architecture for the CGI client to the OLTP system. The alternative embodiment of Figure 7(b) is provided to off-load the OLTP processing from the web server because web servers can be extraordinarily busy responding thousands of clients over the World Wide Web. In this case, the CGI programs which execute OLTP calls have those calls trapped by the Client RPC 305 which is the same remote procedure call technique described above with respect to Figure 3. There is then introduced another machine to operate as a "pseudo-CGI client" 750 which includes in its system the Client RPC 310 which communicates with the Client RPC 305 and forwards the calls to OLTP service core 760 which then provides the OLTP interaction to either a Dispatcher or other OLTP service processors in the system.

In accordance with this aspect of the present invention, extensions to HTML are provided for making appropriate calls for transaction message passing to the OLTP

system. The HTML extensions are implemented by embedding the extension controls in HTML comment code.

The following describes the syntax of the language extensions to HTML for use with the OLTP systems in accordance with an implemented embodiment of the present inventions:

Reserved Internal Setup Variables

A few variables can be set in HTML that control the way the OLTP client processes the HTML document. These variables can be set using the HTML syntax:

<INPUT TYPE="HIDDEN" NAME="IN_WEB_OUTPUT_PAGE"
VALUE="FILENAME.HTM">

Control Variables

IN_WEB_OUTPUT_PAGE - Use the specified files as the template file.

IN_MODE - If the mode is set to "dummy," the transaction will not actually be called and the default value for each variable will be inserted into the template file instead of the OLTP service output.

OLTP Service Variables

The variable DE_TRANSACTION_ID contains the name of the OLTP service to be called. Any variable name sent to the OLTP

client that starts with DE_ will be sent to the specified OLTP service.

DE_TRANSACTION_ID - Name of the OLTP service to be called after the input variables have been set.

Extension Format

The HTML format for a comment is of the form <!-- to begin the comment followed by the sequence--> to end the comment. For example:

<!-- This is a comment within HTML -->

The special OLTP extended command set resides within the HTML comment structure. OLTP commands are preceded by a begin delimiter <!--${and are terminated by an end delimiter }--> as shown below:

<!--${VARIABLE_NAME}-->

The term "OLTP command" is used herein to specify the part of the HTML comment between the ${and the}, which, in the above example, would be VARIABLE_NAME. The OLTP command has two forms:

The single command form (shown above), which specifies only the name of the variable to be substituted at the location of the command in the document.

A longer form allows multiple commands to be performed at the same time.

The short form of the example above is equivalent to the long form of the command:

```
<!--${FIELD=VARIABLE_NAME}-->
```

Valid OLTP Commands

All commands may be shortened to the least number of characters that would keep the command unambiguous. For example, the command FIELD can be shortened to F since there are no other commands that start with the letter "F". Ambiguous commands are accepted on a first-come first-served basis; in other words, the first internal command that provides a match is used.

Commands are specified in the following form:

```
COMMAND=VALUE [COMMAND=VALUE [...]]
```

The following Table I describes the current set of supported commands and associated values:

COMMAND	VALUE	DESCRIPTION
Field	Char string used as a valid variable name to indicate either an input variable or a variable from the service output.	Insert the value of the specified variable into the HTML text.
Type	Numeric Date	
Source	Input	Set this variable from the URL input variables rather than from the OLTP service return values.
Value	Char string	A value to insert into the HTML text if no value for this variable was returned from the OLTP service.
Width	Int	Output width. The output is padded with spaces to the width specified.
Repeat	Line	The current HTML line will repeat until the first variable in the row is not found in the output from the OLTP service. Normally used for creating tables.

TABLE I

With this extension mechanism, an HTML designer can develop HTML pages for interacting with a web browser without requiring the programming skills required for CGI programming. A CGI programmer may separately interact with the HTML pages 720 as shown in Figure 7(a) and 7(b) and extract those embedded commands from the HTML page, execute the OLTP transaction, and return the data into the HTML page. With this mechanism, an HTML designer can design all aspects of the presentation while putting the special comment fields where result data from an OLTP transaction are needed to be inserted using the special variable names. The CGI program will read the HTML page and pass it back to the web browser intact but operating on the embedded commands in the manner described above with respect to the operation of the overall OLTP system.

In simple summary, when data is filled in by a user through a web browser which is exhibiting an HTML page, the DE_ variables are established and routed back to the web server which then provides the HTML page to the CGI script. The CGI script sees the DE_ variables and generates the various OLTP.Put requests to the message object and then sends the message object to the named OLTP service identified by the DE_TRANSACTION_ID. The result is then returned and inserted into the appropriate space specified in the HTML page for result data. This provides a robust OLTP environment with all the advantages of the World Wide Web wherein a client application is never required to be changed by the OLTP system administrators. Only changes in the OLTP service applications need be effected to change the operation seen by the end user.

The preceding describes numerous enhancements to distributed computing architectures for handling on-line transaction processing. Although the present invention has been described in terms of exemplary and implemented embodiments, it will be understood by those of ordinary skill in the art that many of the techniques and aspects of the present invention are suitable for use in other computing contexts. Accordingly, the spirit and scope of the present invention should be measured in terms of the claims which follow.

CLAIMS

What is claimed is:

1. A computer system for providing an interface between a remote client application and an on-line transaction processing (OLTP) system, said computer system comprising:

server logic coupled to communicate with said remote client for receiving data for use in an OLTP transaction request;

at least a first hypertext markup language (HTML) file coupled to be accessed by said server logic for delivery to said remote client, said HTML file providing rendering instructions to said remote client application for presentation to an end user, said HTML file providing for at least a first data input from said client application to be returned to said server logic; and

CGI/OLTP interface processing logic coupled to communicate with said server logic to receive data representing said HTML file and said first data input, said interface processing logic for generating from said HTML file and said first input data an OLTP transaction request to be provided to said OLTP system.

2. The computer system of claim 1 wherein said computer system comprises a World Wide Web (web) server and wherein said client application comprises a web browser for communicating with said web server over a network.

3. The computer system of claim 1 wherein said interface processing logic includes means for inserting into

said HTML file result data received from said OLTP system in response to said transaction request.

4. The computer system of claim 3 wherein said interface processing logic includes means for detecting OLTP transaction request information embedded between comment delimiters in said HTML file.

5. The computer system of claim 4 further comprising first application interface logic for trapping external OLTP calls generated by said CGI/OLTP interface processing logic and for forwarding said calls to a remote pseudo-CGI client of said OLTP system.

6. The computer system of claim 4 wherein said OLTP transaction request is generated by creating a message object which includes data type information with said input data for interpretation by said OLTP system.

7. A method of utilizing a World Wide Web (web) server as a client service to an on-line transaction processing (OLTP) system comprising the steps of:

presenting a hypertext markup language (HTML) page to a remote web browser client application, said HTML page providing at least a first field to receive input data from the web browser;

receiving said input data from said web browser client;

parsing said HTML page to extract said input data for use in an OLTP transaction request;

generating an OLTP service call incorporating said input data;

receiving result data from said OLTP service call; and

forwarding said result data to said web browser client with said HTML page.

8. The method according to claim 7 wherein said parsing step comprises the steps of:

detecting an HTML comment field in said HTML page;

detecting an OLTP system instruction delimiter in said comment field; and

extracting the information identified by said delimiter.

9. The method according to claim 8 wherein said forwarding step comprises the step of inserting said result data into said HTML page at a point indicated by an OLTP instruction embedded in said HTML page.

10. A method of utilizing an HTML document to convey non-HTML information comprising the steps of:

embedding said non-HTML information in an HTML comment field in said document;

detecting said embedded information;

extracting said embedded information from said HTML document; and

forwarding said embedded information to a destination configured to utilize said embedded information.

11. The method according to claim 10 wherein said embedding step comprises the steps of:

embedding begin and end delimiters in said HTML comment field; and

inserting said non-HTML information between said begin and end delimiters in said comment field.

12. The method according to claim 11 wherein said detecting step comprises the step of detecting said begin and end delimiters in said comment field.

13. The method according to claim 12 wherein said extracting step includes the step of extracting embedded information from between said begin and end delimiters.

14. A method of remotely operating a first application running on a first computer which presents procedure calls to a second application running on a second computer comprising the steps of:

detecting a procedure call to said second application;

trapping said procedure call with a first interface application;

establishing a communication path to a second interface application operating on said second computer; and

forwarding said procedure call over said communication path to said second interface application wherein said second interface application causes said procedure call to be initiated from said second computer.

15. A method of passing data from a first application running on a first computer system operating in accordance with a first platform to a second application running on a second computer in accordance with a second platform comprising the steps of:

creating a message object;

putting a variable and its associated value into said message object;

putting into said message object the data type of said variable and length of said associated value; and

sending said message object to said second application operating on said second computer system.

16. A method for a dispatcher system in an OLTP system to minimize the bandwidth utilized between a client application and services associated with said OLTP system comprising the steps of:

receiving a transaction request from said client application wherein said transaction request implicates invocation of a plurality of said services in said OLTP system;

referring to a service table to determine which of said OLTP services to invoke in response to said transaction request; and

invoking the OLTP services identified in said service table as corresponding to said transaction request.

17. The method according to claim 16 further comprising the steps of:

grouping any response information resulting from invoking said OLTP services; and

providing the grouped response information to said client application.

FIG. 1

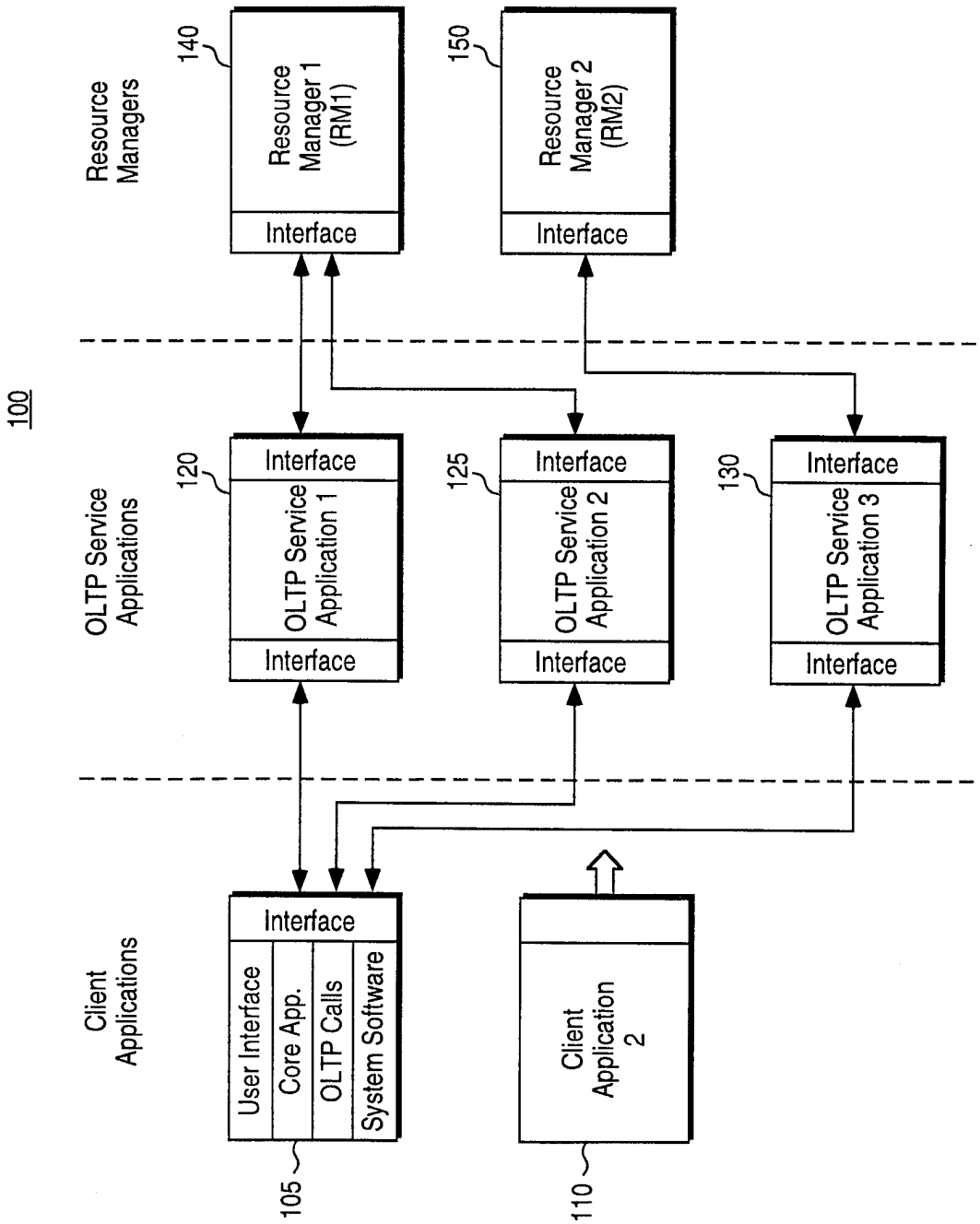

100
Resource Managers
140
Resource Manager 1 (RM1)
Interface
150
Resource Manager 2 (RM2)
Interface
OLTP Service Applications
120
Interface
OLTP Service Application 1
Interface
125
Interface
OLTP Service Application 2
Interface
130
Interface
OLTP Service Application 3
Interface
Client Applications
Interface
User Interface
Core App.
OLTP Calls
System Software
105
Client Application 2
110

FIG. 2

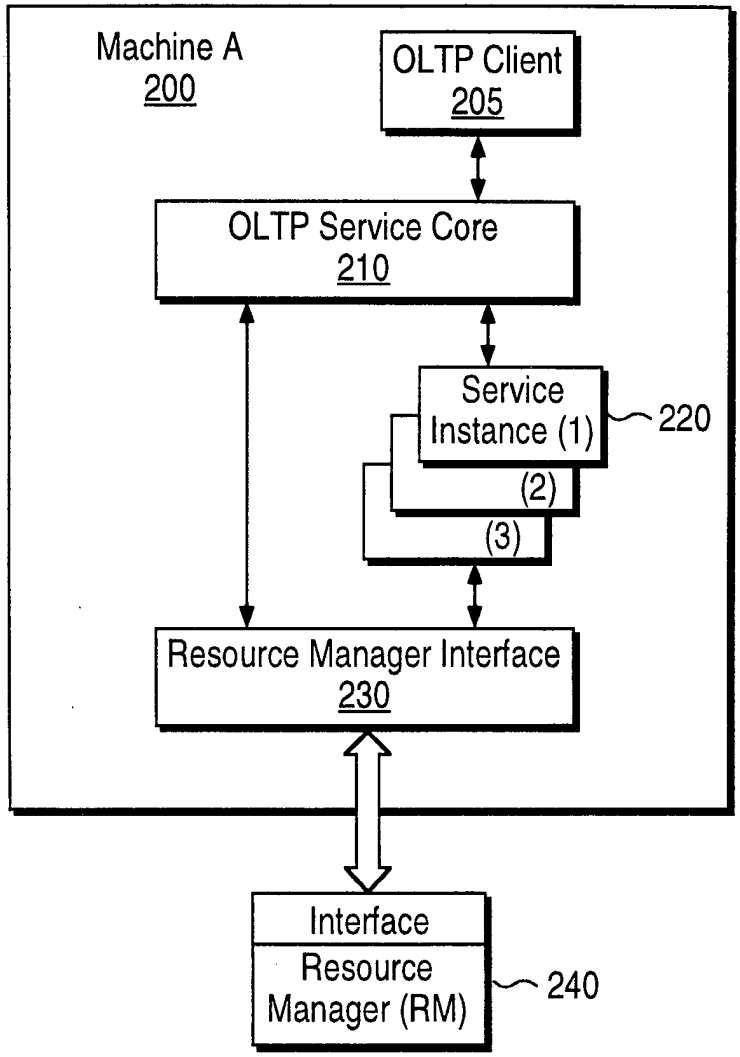
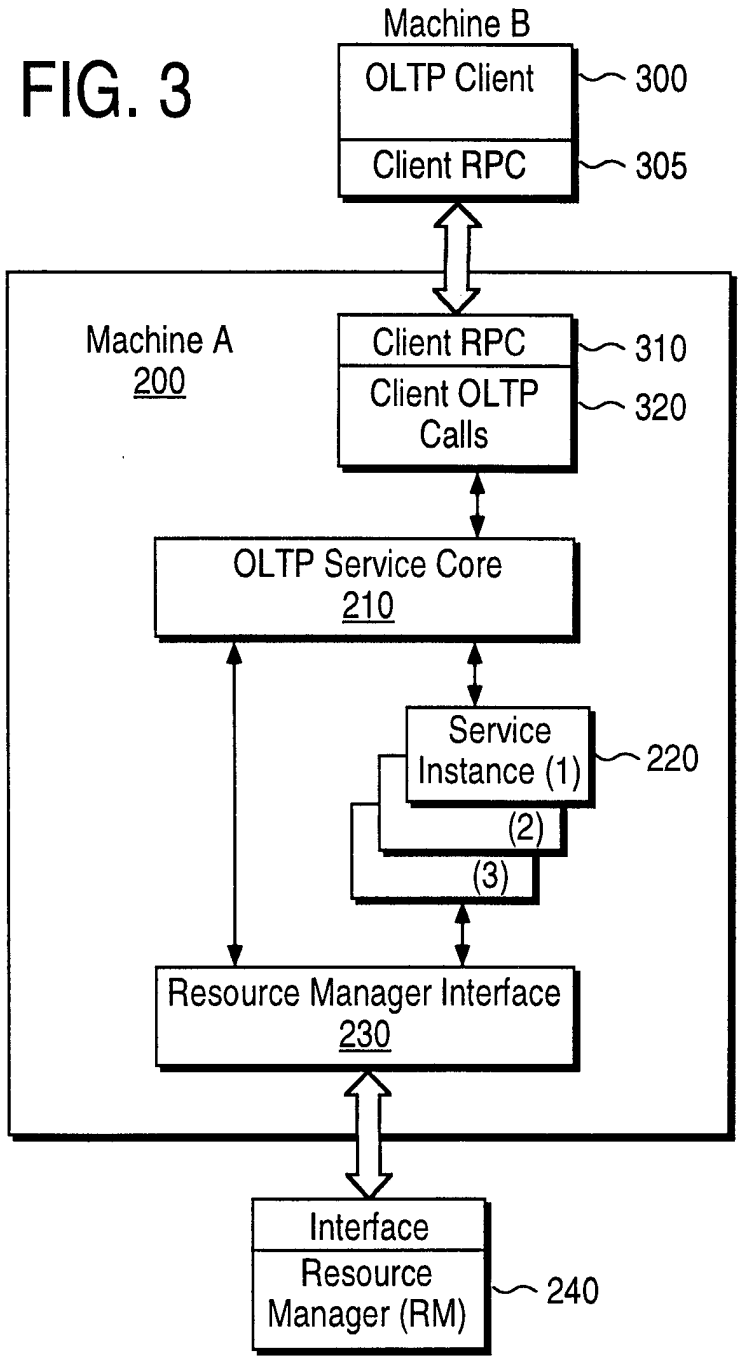

Machine A
200
OLTP Client
205
OLTP Service Core
210
Service
Instance (1)
(2)
(3)
220
Resource Manager Interface
230
Interface
Resource
Manager (RM)
240

FIG. 3


Machine B
OLTP Client
300
Client RPC
305
Machine A
200
Client RPC
310
Client OLTP Calls
320
OLTP Service Core
210
Service Instance (1)
220
(2)
(3)
Resource Manager Interface
230
Interface
Resource Manager (RM)
240

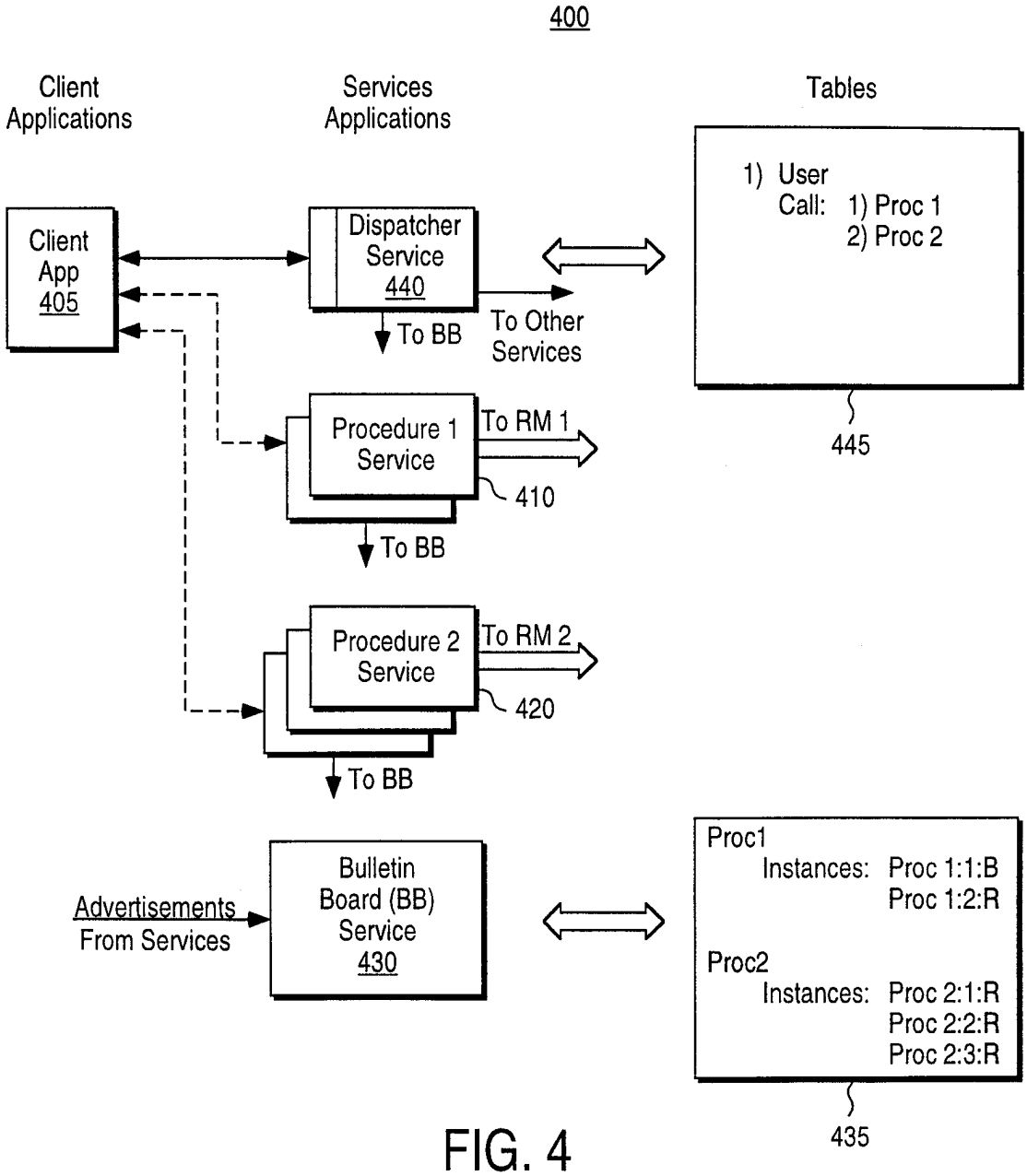

400
Client Applications
Services Applications
Tables
Client App 405
Dispatcher Service 440
To BB
To Other Services
1) User
Call: 1) Proc 1
2) Proc 2
445
Procedure 1 Service
To RM 1
410
To BB
Procedure 2 Service
To RM 2
420
To BB
Advertisements From Services
Bulletin Board (BB) Service 430
Proc1
Instances: Proc 1:1:B
Proc 1:2:R
Proc2
Instances: Proc 2:1:R
Proc 2:2:R
Proc 2:3:R
435

FIG. 4

FIG. 5

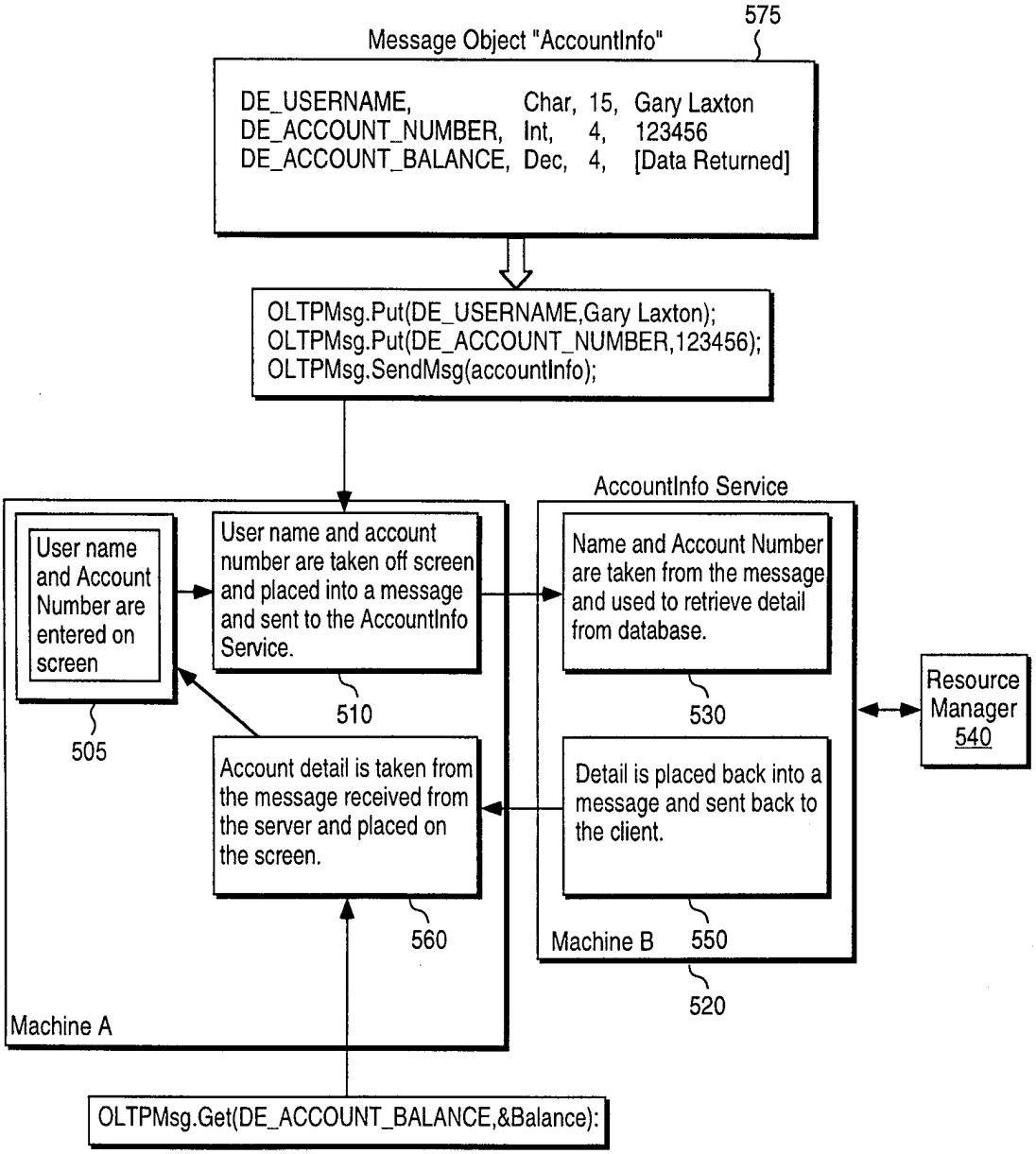

575
Message Object "AccountInfo"
DE_USERNAME, Char, 15, Gary Laxton
DE_ACCOUNT_NUMBER, Int, 4, 123456
DE_ACCOUNT_BALANCE, Dec, 4, [Data Returned]
OLTPMsg.Put(DE_USERNAME,Gary Laxton);
OLTPMsg.Put(DE_ACCOUNT_NUMBER,123456);
OLTPMsg.SendMsg(accountInfo);
AccountInfo Service
User name and Account Number are entered on screen
505
User name and account number are taken off screen and placed into a message and sent to the AccountInfo Service.
510
Name and Account Number are taken from the message and used to retrieve detail from database.
530
Resource Manager 540
Account detail is taken from the message received from the server and placed on the screen.
560
Detail is placed back into a message and sent back to the client.
Machine B 550
520
Machine A
OLTPMsg.Get(DE_ACCOUNT_BALANCE,&Balance);

FIG. 6

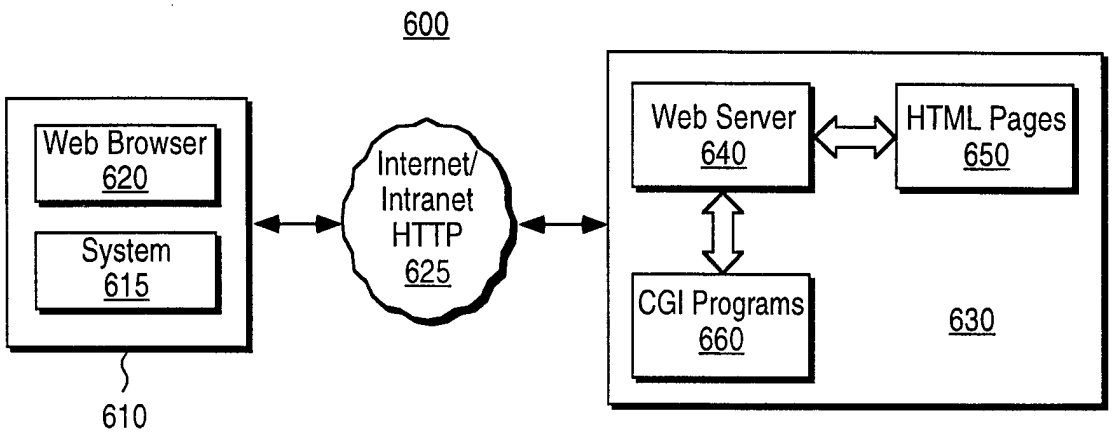

600
Web Browser
620
System
615
610
Internet/
Intranet
HTTP
625
Web Server
640
HTML Pages
650
CGI Programs
660
630

FIG. 7(a)

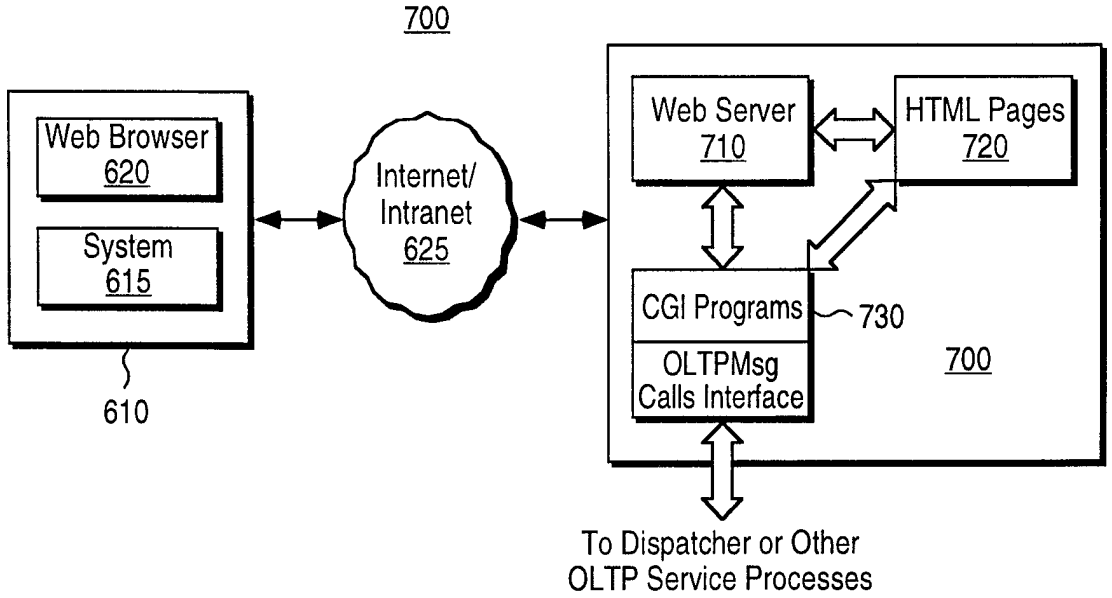
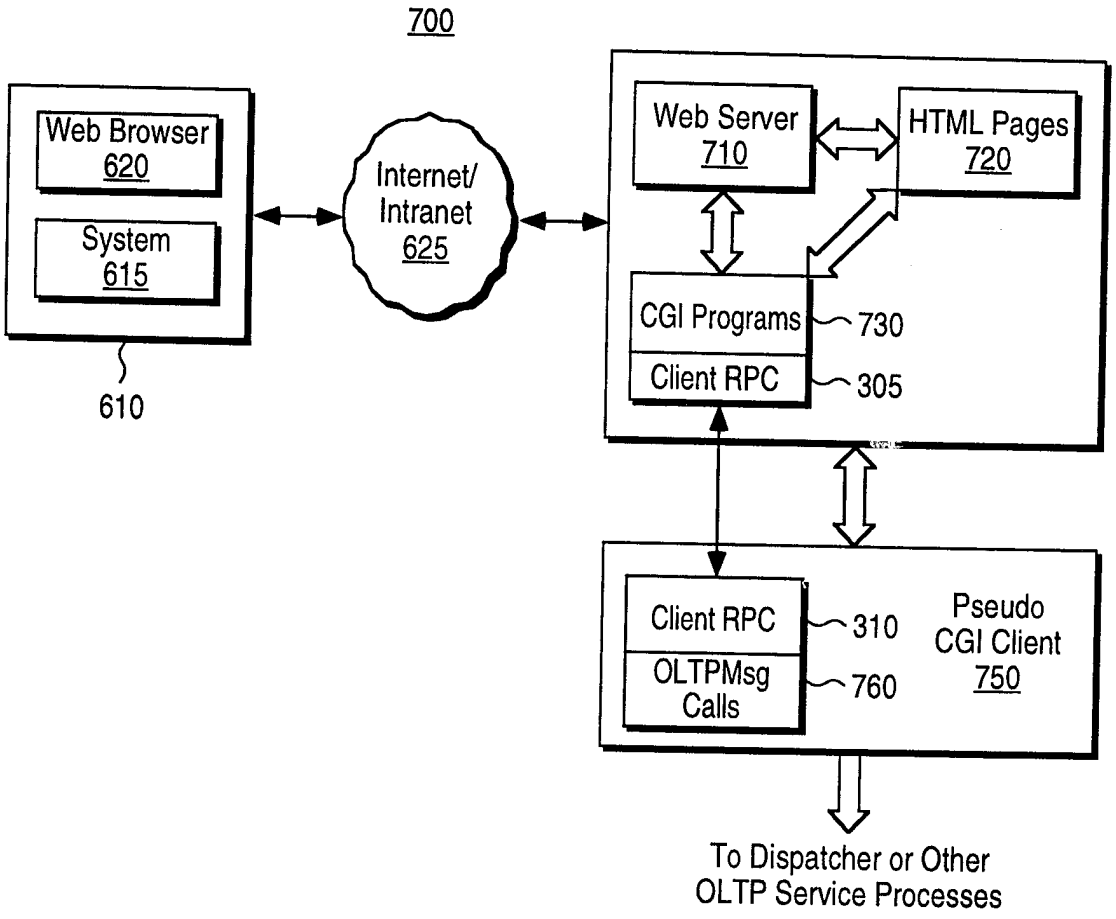

700
Web Browser
620
System
615
610
Internet/
Intranet
625
Web Server
710
HTML Pages
720
CGI Programs
730
OLTPMsg
Calls Interface
700
To Dispatcher or Other
OLTP Service Processes

FIG. 7(b)


700
Web Browser
620
System
615
610
Internet/
Intranet
625
Web Server
710
HTML Pages
720
CGI Programs
730
Client RPC
305
Client RPC
310
OLTPMsg
Calls
760
Pseudo
CGI Client
750
To Dispatcher or Other
OLTP Service Processes