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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
17 April 2003 (17.04.2003)

PCT

(10) International Publication Number
WO 03/032181 A1

(51) International Patent Classification[7]: G06F 15/16

(21) International Application Number: PCT/US02/31727

(22) International Filing Date: 4 October 2002 (04.10.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/327,530	5 October 2001 (05.10.2001)	US
10/264,973	3 October 2002 (03.10.2002)	US

(71) Applicant: **BEA SYSTEMS, INC.** [US/US]; 2315 North First Street, San Jose, CA 95131 (US).

(72) Inventors: **MESSINGER, Adam**; 317 29th Street #306, San Francisco, CA 94131 (US). **PULLARA, Sam**; 2030 3rd Street #14, San Francisco, CA 94107 (US). **BROWN, Dave**; 1441 Montgomery Street, Apartment 3, San Francisco, CA 94133 (US).

(74) Agents: **MEYER, Sheldon, R.** et al.; Fliesler Dubb Meyer and Lovejoy LLP, Four Embarcadero Center - Suite 400, San Francisco, CA 94111-4156 (US).

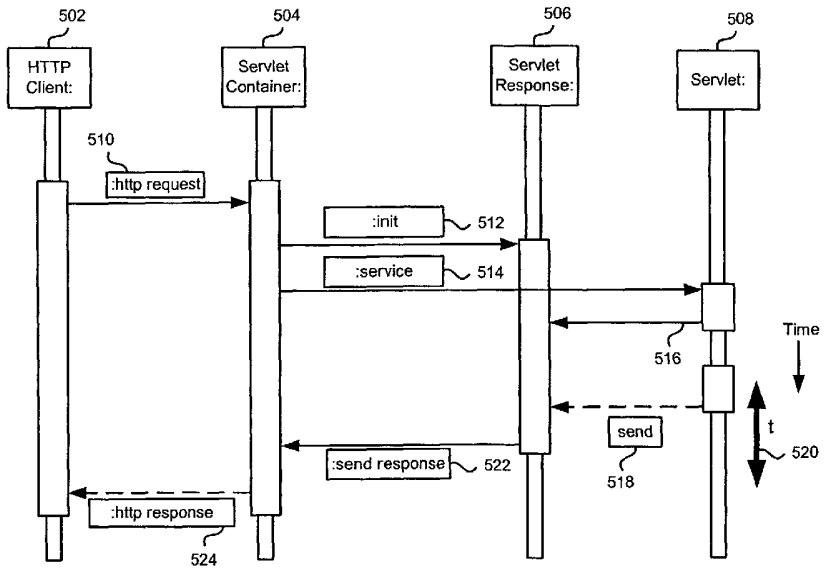
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT,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

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

Published:
— *with international search report*

[Continued on next page]

(54) Title: SYSTEM FOR INTEGRATING JAVA SERVLETS WITH ASYNCHRONOUS MESSAGES



(57) Abstract: In a traditional server that users servlets, when a rHTTP client (502) request is dispatched to a thread the service method of the appropriate servlet (508) is called. When the service method returns, the response is sent. This is sub-optimal in the case that an asynchronous event must occur before the response can be sent, because the thread running the servlet must block until the event occurs. The invention provides for asynchronous processing of such requests. In one embodiment, the invention provides an extension to the Servlet API which allows the server method to return and thus the thread to be freed before the response is ready to be sent. Then when the asynchronous event later occurs the response may be completed and sent.

WO 03/032181 A1

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

SYSTEM FOR INTEGRATING JAVA SERVLETS WITH ASYNCHRONOUS MESSAGES

Claim of Priority:

[0001] This application claims priority from provisional application "SYSTEM FOR APPLICATION SERVER MESSAGING WITH ASYNCHRONOUS QUEUES", Application No. 60/327,530, filed October 5, 2001, and which application is incorporated herein by reference.

Field of the Invention:

[0002] The invention relates generally to application and transaction servers and particularly to a system for supporting message queuing and threads with multiple execute queues.

Cross References:

[0003] This application is related to provisional application "SYSTEM FOR APPLICATION SERVER MESSAGING WITH MULTIPLE DISPATCH POOLS", Application No. 60/327,543, filed October 5, 2001, and Utility Patent Application "SYSTEM FOR

APPLICATION SERVER MESSAGING WITH MULTIPLE DISPATCH POOLS", Application Number _____, Inventors: Adam Messinger and Don Ferguson, filed October 3, 2002 both applications are incorporated herein by reference.

Background of the Invention:

[0004] The Java 2 Platform, Enterprise Edition (J2EE) specification defines one of the current standards for developing multi-tier enterprise applications. J2EE provides a component-based approach to the design, development, assembly, and deployment of enterprise applications, which both reduces the cost and enables fasten design and implementation. The J2EE platform gives the developer a multi-tiered distributed application model, the ability to reuse components, a unified security model, and flexible transaction control. Not only can they deliver innovative customer solutions to market faster than ever, but the resultant platform-independent J2EE component-based solutions are not tied to the products and application program interfaces (APIs) of any one vendor.

[0005] The J2EE specification defines the following kinds of components: application client components; Enterprise JavaBeans (EJB); servlets and Java Server Pages (JSP) (also called Web components); and applets. A multi-tiered distributed application model implies that the application logic is divided into components according to function, and different application components may make up a J2EE application on the same or different servers. Where an application component is actually installed depends on which tier in the multi-tiered J2EE environment the application component belongs. These tiers are

depicted in **Figure 1.** As shown therein an application server tier **104** is used to develop EJB containers and/or presentation containers such as servlets, JSP, and html pages **114**. These in turn are used as an interface between a client tier **102**, where the clients **108** and client applications are deployed, and a backend tier **106**, used for hosting enterprise or legacy applications such Enterprise Resource Planning (ERP) systems.

[0006] Client tier - These can be browsers, Java-based programs, or other Web-enabled programming environments running within the client tier, both inside and outside of corporate firewalls.

[0007] Application Server tier - Normally this tier hosts a combination of presentation logic and business logic to support client requests. Presentation logic is supported via JSP pages and servlets that display HTML pages, while business logic is supported via Remote Method Invocation (RMI) objects and EJBs **112**. EJBs rely upon the container environment for transactions, lifecycle and state management, resource pooling, security, etc., which together make up the run time environment in which the beans are executed.

[0008] Back-end tier - This is generally a combination of existing applications and data stores. It is also referred to as the Enterprise Information Systems (EIS) tier, since it may include such systems as Enterprise Resource Planning (ERP), mainframe transaction processing, database systems, and other legacy information systems.

[0009] Since the components of a J2EE application run separately, and often on different devices, there needs to be a way for client and application server tier code to look up and reference other code and resources. Client and application code can, for example, use

the Java Naming and Directory Interface (JNDI) **116** to look up user-defined objects such as enterprise beans, and environment entries such as the location of the Java Database Connector (JDBC) DataSource objects, which in turn are used for looking up resources in backend tier, and message connections.

[0010] Application behavior such as security and transaction management can be configured at deployment time on Web and enterprise bean components. This deployment time feature decouples application logic from the configuration settings that might vary with the assembly. The J2EE security model lets a developer configure a Web or enterprise bean component so that system resources are accessed only by authorized users. For example, a Web component can be configured to prompt for a user name and password. An Enterprise Bean component can be configured so that only persons in specific groups can invoke certain kinds of its methods. Alternatively, a servlet component might be configured to have some of its methods accessible to everyone, and a few methods accessible to only certain privileged persons in an organization. The same servlet component can be configured for another environment to have all methods available to everyone, or all methods available to only a select few.

[0011] Some application servers, such as the WebLogic Server product from BEA Systems, Inc., San Jose, California, use an Access Control List (ACL) mechanism that allows for fine-grained control of the usage of components running on the server. Using an ACL, a developer can define at the Java Method level what can, or cannot, be executed by which user or group of users. This ACL mechanism covers anything that runs on the application server except for EJBs, which have their

own access control mechanism defined in the EJB specification. Security realms allow the administrator to import information from existing authorization or authentication systems into the ACL.

Java Servlets

[0012] A servlet is a program that extends the functionality of a Web server. A servlet receives a request from a client, dynamically generates the response (possibly querying databases to fulfill the request), and then sends the response containing an HTML or XML document to the client. Servlets are similar to CGI but are typically easier to write, since servlets use Java classes and streams. They execute faster because servlets are compiled to Java byte code and at run time the servlet instance is kept in memory, each client request spawning a new thread. Servlets make it easy to generate data to an HTTP response stream in a dynamic fashion. Each client request is performed as a new connection, so flow control does not come naturally between requests. To allow for this session management maintains the state of specific clients between requests. In some application servers, servlets make use of the HTTP session object to save their state between method requests. This object can be replicated in a clustered environment for failover purposes.

Java Server Pages

[0013] JSP pages are a text-based, presentation-centric way to develop servlets. JSP pages offer all the benefits of servlets, and when combined with a JavaBeans class, provide an easy way to keep content and display logic separate. Both JSP pages and servlets are more

desirable than Common Gateway Interface (CGI), because they are platform-independent, and use less overhead. JSP pages can be used with JavaBeans classes to define Web templates for building a Web site made up of pages with a similar look and feel. The JavaBeans class performs the data rendering, so the templates have no Java code. This means they can be maintained by an HTML editor. Simple Web-based application using a JSP page can be used to bind content to application logic using custom tags or scriptlets instead of a JavaBeans class. Custom tags are bundled into tag libraries that are imported into a JSP page. Scriptlets are small Java code segments embedded directly in the JSP page.

Java Messaging Services (JMS)

[0014] JMS is the J2EE mechanism used to support the exchange of messages between Java programs. This is how Java supports asynchronous communication, wherein the sender and receiver don't need to be aware of each other and thus can operate independently. JMS supports two messaging models:

[0015] Point to point - which is based on message queues. In this model message producer sends a message to a queue. A message consumer can attach itself to a queue to listen for messages. When a message arrives on the queue, the consumer takes it off the queue and responds to it. Messages can be sent to just one queue and will be used by just one consumer. Consumers have the option to filter messages to specify the exact message types they want.

[0016] Publish and subscribe - which allows producers to send messages to a topic and for all the registered consumers for that topic

to retrieve those messages. In this case, many consumers can receive the same message.

[0017] One problem with current Servlet APIs is the completely synchronous programming model. After a request is dispatched to a particular thread the service() method of the appropriate servlet is called. When the service() method returns, the response is sent. This is a simple programming model which is suitable for many types of work, but is sub-optimal in the case that a asynchronous event must occur before the response can be sent, because the thread running the servlet must block until the event occurs.

Summary of the Invention:

[0018] The invention provides a system and method for asynchronous threading which allows the service() method to return (and thus allowing the thread to be freed up) before the response is ready to be sent. Then when the asynchronous event later occurs the response may be completed and sent. An example use of this mechanism is the use of JMS in conjunction with servlets.

[0019] In accordance with the invention, the process begins when a servlet is executed. The servlet builds a portion of a response, but typically needs more data to complete the response. While it's waiting it queues a JMS message requesting the data and sets the response object aside in a place where it may be found when a JMS message containing the needed data arrives. At this point the servlet may return, but the response will not yet be sent. At a later point in time, when the data arrives via JMS for example, the corresponding response object is

retrieved. The remainder of the response can then be generated. When the response is completed it can be explicitly sent to the client.

[0020] This feature is also available through the use of a JSP tag library. Using the tags the JSP page author specifies what work should be done before the asynchronous event and which work should be done after the asynchronous event. This feature integrates with the JSP context mechanisms to ensure that they are restored after the asynchronous event and that processing can continue uninterrupted.

Brief Description of the Figures:

[0021] **Figure 1** shows an illustration of a J2EE compatible architecture that can utilize the present invention.

[0022] **Figure 2** shows an illustration of a threading policy with asynchronous thread pool in accordance with an embodiment of the invention.

[0023] **Figure 3** shows a diagram of a synchronous threading process.

[0024] **Figure 4** shows a lifecycle of a single HTTP request that is processed using traditional methods.

[0025] **Figure 5** shows a lifecycle of a single HTTP request processed using asynchronous messaging.

[0026] **Figure 6** shows a lifecycle of a plurality of HTTP requests processed using traditional methods.

[0027] **Figure 7** shows a lifecycle of a plurality of HTTP requests processed using asynchronous messaging.

Detailed Description:

[0028] Broadly described, the invention provides a system and method to allow asynchronous threading. The invention can be incorporated into application server systems that allow access to a servlet via an Application Program Interface (API), or into other systems that benefit from asynchronous threading.

[0029] The typical Servlet APIs are completely synchronous. After a request is dispatched to a thread, the service() method of the appropriate servlet is called. When the service() method returns, the response is sent. This simple programming model is suitable for many types of work, but is sub-optimal in those instances that an asynchronous event must occur before the response can be sent, because the thread running the servlet must block until the event occurs.

[0030] In one embodiment, the invention provides an extension to the Servlet API which allows the service() method to return (and thus allowing the thread to be freed up) before the response is ready to be sent. Then when the asynchronous event later occurs the response may be completed and sent. One example use of this mechanism is the use of JMS in conjunction with servlets.

[0031] In this embodiment, when a servlet is executed, it builds a portion of a response, but then typically needs more data to complete the response. It queues a JMS message requesting the data, and sets the response object aside in a place where it may be found when a JMS message containing the needed data arrives. At this point the servlet may return, but the response will not yet be sent. Later on, when the required data arrives via JMS, the response object is retrieved. The

remainder of the response can then be generated, and when completed can be explicitly sent to the client.

[0032] The invention is primarily designed for use with application, transaction, and messaging servers, such as the WebLogic family of products from BEA Systems, Inc. At the core of the typical server's design is the threading model, the policy by which threads are assigned to perform work requests. As servlet requests arrive at the server they are dispatched to a thread. This thread is responsible for executing the requested servlet. The server employs a threading model which uses two thread pools an asynchronous pool (often referred to as reader threads) and a synchronous pool (referred to as execute threads). This combination of pools allows a developer or administration to effectively prioritize requests while tolerating user code that performs blocking operations.

[0033] **Figure 2** shows a threading policy mechanism **206** in accordance with an embodiment of the invention. The asynchronous thread pool **208** waits on an asynchronous input mechanism **202** (muxer) for asynchronous read results to become available. Once a result is available a thread from the pool looks at the message and dispatches it by making the appropriate callbacks. The dispatch callbacks usually queue the request for later processing by the synchronous thread pool. However certain non-blocking, priority requests are services directly in the callback. By aggressively accepting input high priority requests **214** do not wait to be read while low priority requests **212** run. Since these threads should never block there are usually a low number of them, perhaps one per processor (CPU).

[0034] The synchronous thread pool **210** waits on a queue of requests **204**. Once a request is available a thread from the pool processes takes the request from the queue, processes it, and sends out the result **216**. While processing the request the thread may execute code, such as sending out the result, which causes the thread to block. The number of threads should therefore be tuned so that there is always one thread per CPU that is in the runnable state. The dispatch policies are described in more detail in provisional application entitled, "SYSTEM FOR APPLICATION SERVER MESSAGING WITH MULTIPLE DISPATCH POOLS", Application Number 60/327,543, Inventor: Adam Messinger, filed October 5, 2001 and copending utility application entitled, "SYSTEM FOR APPLICATION SERVER MESSAGING WITH MULTIPLE DISPATCH POOLS", Application Number _____, Inventors: Adam Messinger and Don Ferguson, filed October 3, 2002.

[0035] **Figure 3** shows a traditional synchronous message response mechanism. As shown therein, a request from the client application **302**, such as for example a Web browser application, is transmitted to the application server via a servlet **304**. The request may be in the form of a hypertext transmission protocol (http) request **306**, for which the client will typically expect a hypertext markup language (html) response **308**. In the synchronous model the thread executes the servlet and then immediately sends the response to the client when execution of the servlet completes. The problem with this approach is that the executing thread is consumed for the entire execution of the servlet. If the servlet is performing tasks which block, perhaps waiting for other data, then this can represent a waste of server resources.

[0036] **Figure 4** illustrates a typical system lifecycle wherein a client access a resource at a server. As shown in **Figure 4**, an HTTP client **402** accesses a servlet **408**, which typically runs on a remote web server. It will be evident to one skilled in the art that while HTTP clients are shown herein for purposes of illustration, the invention is not so limited, but may be used with other types of client application. As shown in the lifecycle diagram in **Figure 4**, the HTTP client accesses the servlet via a servlet container **404**. The servlet container is responsible for receiving the HTTP request **410**, and passing it to the servlet **408** for processing. Much of the operation of processing this HTTP request takes place at the servlet response level **406**. As illustrated in **Figure 4**, with time increasing vertically down the page, the process continues with an `:init` call **412** to the servlet, which is handled by the response handler **406**. The servlet container then passes a `:service` request **414** to the servlet, to retrieve or to update data for example. A typical use of such a system is in an e-commerce environment, wherein the client application is designed to retrieve catalog listings, such as the results of a search for flight times, etc. The servlet typically responds to the request by writing output **416** to the servlet response handler. This step is often required for buffering, and for optimization purposes. When the servlet container then requests that the response be returned to the client, it sends a `:send` request **418** to the response handler, and the response handler returns the `:send response` **420** to the servlet container. The response is then sent as an HTTP response **422** to the client.

[0037] **Figure 5** illustrates a similar life cycle that may be used in accordance with an embodiment of the invention. As shown in **Figure**

5, again an HTTP client **502** is used to access a remote server, server resource or servlet **508**. The HTTP request **510** is handled by a servlet container **504** which issues an `:init` request **512** to the servlet response handler **506**. This time however, when the `:service` request **514** is transmitted to the servlet for processing, the servlet returns **516** immediately to the response handler. This immediate return frees up the response handler for handling subsequent requests, in that it does not need to wait for the servlet to actively return data in order to handle those requests. After a period of time t **520**, when the servlet has the appropriate data to return to the requests, it sends a `:send` signal **518** to the response handler, which then sends the `:send response` **522** to the servlet container. The subsequent HTTP response **524** is transmitted to the client as before.

[0038] As part of the process described above, the servlet sets a response code until it has something else to transmit, effectively taking the responsibility for responding away from the container level and placing it at the servlet level. In practice the amount of time that the servlet waits to issue the `:send` response can be defined as some arbitrary amount, or can be performed as the result of an asynchronous message, for example as the result of receiving a JMS message indicating that the information is available to be transmitted to the client. This type of processing is useful in, for example, e-commerce sites where a user typically experiences a delay time in awaiting search results. When processing is performed according to the invention, instead of merely having a frozen screen, the user may receive some items of information, while other items are returned piecemeal as the servlet finds the appropriate data and returns it. At the same time the

servlet response handler is available to handle other client requests. The type of data that is returned immediately, and the type that is returned later, can be specified by the developer.

[0039] **Figures 6 and 7** illustrate in more detail the operation of the invention, as it may be applied to service multiple requests. As shown in **Figure 6**, when asynchronous messaging is not used, subsequent requests from clients must be handled in a sequential manner. So, for example, in **Figure 6**, a first HTTP request **410** from client A is handled by the servlet container **404** and servlet response handler **406** and completely processed, prior to a second HTTP request **430** from client B **403** being handled. The overall result is one of taking twice as much time to process HTTP requests from the two clients. If the requests were not handled in this sequential manner, it is very likely that one or more requests would create a backlog for other requests such that the user would experience a delay in processing.

[0040] **Figure 7** illustrates a life cycle of a mechanism in accordance with an embodiment of the invention in which asynchronous messaging is used to process multiple requests from a single client, and/or requests from multiple clients, in an asynchronous manner, such that the processing can be run in different threads. As shown in **Figure 7**, a first HTTP client A **502** and a second HTTP client B **503** access a servlet resource **508** using the mechanisms described above. In accordance with this embodiment, when a first HTTP request A **510** is received at the servlet container, it is handled by the response handler using an `:init A` call **512**, and then passed as a `:service` request **514** to the servlet **508**. The servlet returns **516** immediately to the servlet response handler, which then frees up the response handler

for handling other requests. As shown in **Figure 7**, a second HTTP request B is handled immediately as an `:init B` call **532** by the servlet response handler, and passed to the servlet as a `:service` request **534** for processing. Again, the servlet returns immediately **536**. Interleaving the messages in this manner reduces the overall time for processing both requests, and allows the servlet to return information to the client when and if it becomes available. For example, as shown in **Figure 7**, when the servlet finds the information necessary to respond to request A it returns that `:send response A` signal **522** to the servlet container for sending on to the client as HTTP response A **524**. A second `:send response B` signal **542**, and HTTP response B **544** is similarly handled in the same way.

Implementation

[0041] The file servlet can be replaced by a fast file servlet on platforms that support the asynchronous sending of files over the network. The implementation of this type of servlet requires the addition of asynchronous responses for servlets, which is discussed below.

Synchronous and Asynchronous Responses

[0042] When a servlet request from a remote client is serviced a response is often required. This response can either be synchronous, in that it is sent by the same thread that processed the request, or asynchronous in that it is sent later in a different thread. This analysis is from the server's perspective. From the clients perspective either type of response may or may not block waiting for it depending upon how the remote request was made.

[0043] Currently most requests are handled in a synchronous manner. When a servlet request is serviced all processing must be completed before the servicing thread can move on to another request. This synchronous model is the one specified by the RMI and servlet specifications. The traditional reason for this is that writing asynchronous code is very difficult and thus prone to error.

[0044] There are certain situations where the ability to respond to requests in an asynchronous manner would be very helpful for conserving threads. This is typically true in cases where the server needs to make one or more long running requests of external resources or where the server needs to wait on some condition while processing the request. An example of this is a client request to dequeue from a JMS queue that is currently empty. In a synchronous model the thread servicing the request blocks until there is a message in the queue to return to the client. In an asynchronous model the thread can set the request aside and continue servicing other requests. When a message is placed in the queue the request can be found and a response sent to the client. The invention allows servers to support asynchronous responses to RMI and servlet requests.

Servlets

[0045] Particular servlets can be declared as asynchronous in their deployment descriptor. When the service method of an asynchronous servlet returns, no further action will be taken on that request. The servlet is responsible for storing the request someplace such that after some other action takes place it can be retrieved and the response sent. At this point a special send() method must be called on

the request which will flush the streams, log the request, and, if it is a keep alive connection then register the socket with the muxer to receive more data. It is important to create implementations that ensure resources are appropriately freed by timing out long running requests, thus freeing resources for garbage collection and other cleanup.

JSP

[0046] An asynchronous model may also be supported in a similar manner through the use of JSP tag libraries. These tag libraries are used by the http developer/page author to designate which portion of the web page should be executed prior to the asynchronous event and which portion should be executed after the asynchronous event. The tag libraries allow the author to gain access to an object which should be notified when the asynchronous event occurs and JSP page execution should resume.

[0047] When using the tag libraries the execution context of the page may be automatically stored before registering for the asynchronous event. In this way it is possible to hide many of the details of asynchronous programming from the JSP author. The JSP need not be concerned about state maintenance. State stored in any of the standard scopes (page, request, session or application) will continue to work as they would using a synchronous JSP.

[0048] The foregoing description of the present invention has been provided for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Obviously, many modifications and variations will be apparent to the practitioner skilled in the art. The embodiments were

chosen and described in order to best explain the principles of the invention and its practical application, thereby enabling others skilled in the art to understand the invention for various embodiments and with various modifications that are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalence.

Claims:

What is claimed is:

1. A system for asynchronous messaging between Java servlets and a servlet handler, comprising:

a server including a servlet and a servlet response handler;

an HTTP interface for receiving requests from, and sending responses to an HTTP client; and,

wherein a request from a client allows the servlet to return certain response data immediately, and to set a response code to be used with subsequent responses.

2. The system of claim 1 wherein the servlet subsequent response is triggered by a response command to send additional response data.

3. The system of claim 2 wherein the response command is a JMS message.

4. The system of claim 1 wherein the functioning of the servlet and servlet response handler is exposed to the user via a JSP tag library.

5. The system of claim 4 wherein the JSP tag library is used to indicate which HTTP response data should be returned immediately and which response data should be returned later.

6. The system of claim 4 further comprising support for JSP

execution contexts.

7. A method for asynchronous messaging between Java servlets and a servlet handler, comprising the steps of:

receiving request from an HTTP client to access a servlet resource; and,

responding to said requests asynchronously, wherein a request from a client allows the servlet to return certain response data immediately, and to set a response code to be used with subsequent responses.

8. The method of claim 7 wherein the servlet subsequent response is triggered by a response command to send additional response data.

9. The method of claim 8 wherein the response command is a JMS message.

10. The method of claim 7 wherein the functioning of the servlet and servlet response handler is exposed to the user via a JSP tag library.

11. The method of claim 10 wherein the JSP tag library is used to indicate which HTTP response data should be returned immediately and which response data should be returned later.

12. The system of claim 10 further comprising support for JSP execution contexts.

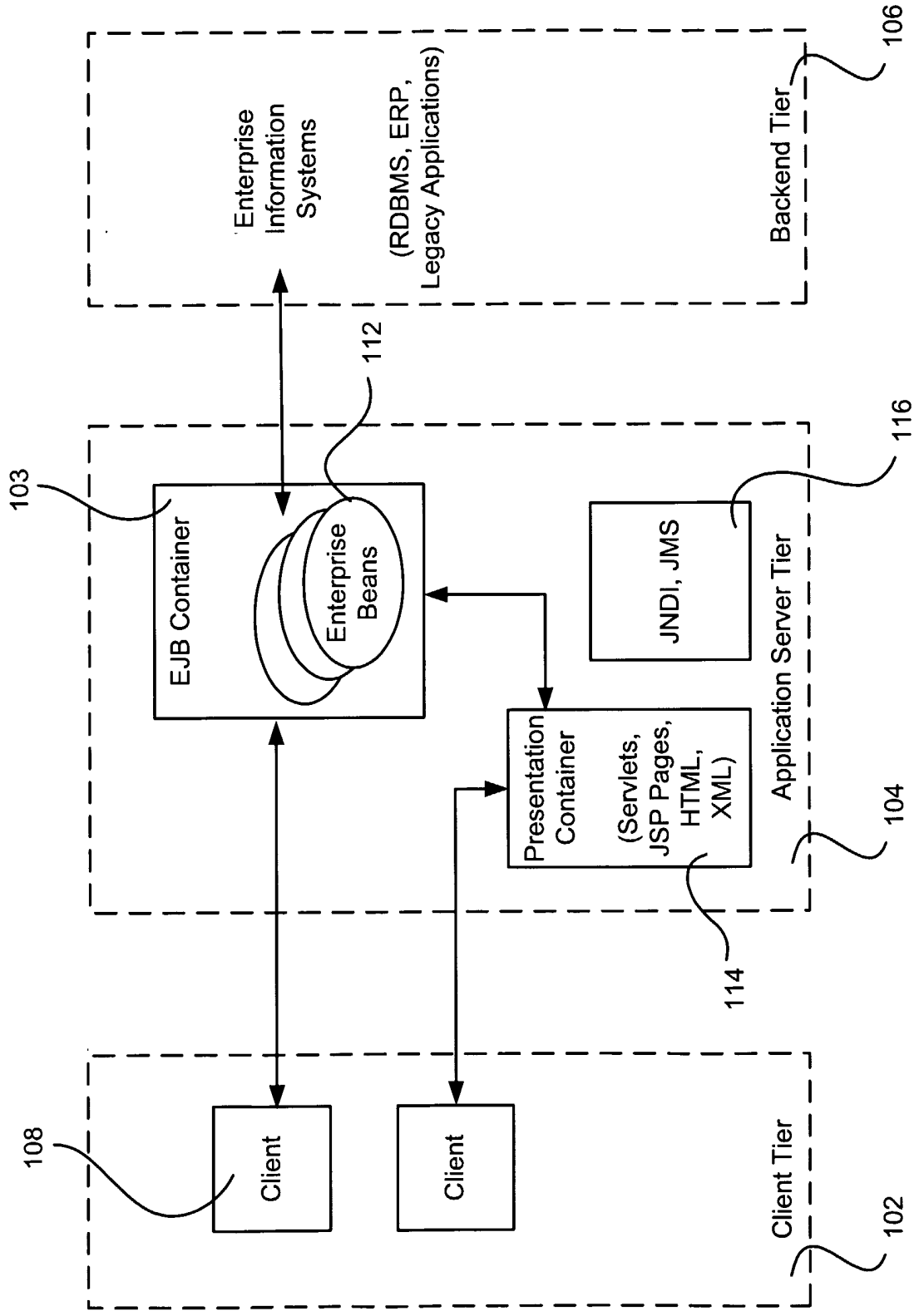

Enterprise Information Systems
(RDBMS, ERP, Legacy Applications)
Backend Tier
106
112
103
EJB Container
Enterprise Beans
JNDI, JMS
116
Presentation Container
(Servlets, JSP Pages, HTML, XML)
Application Server Tier
104
114
108
Client
Client
Client Tier
102

FIGURE 1

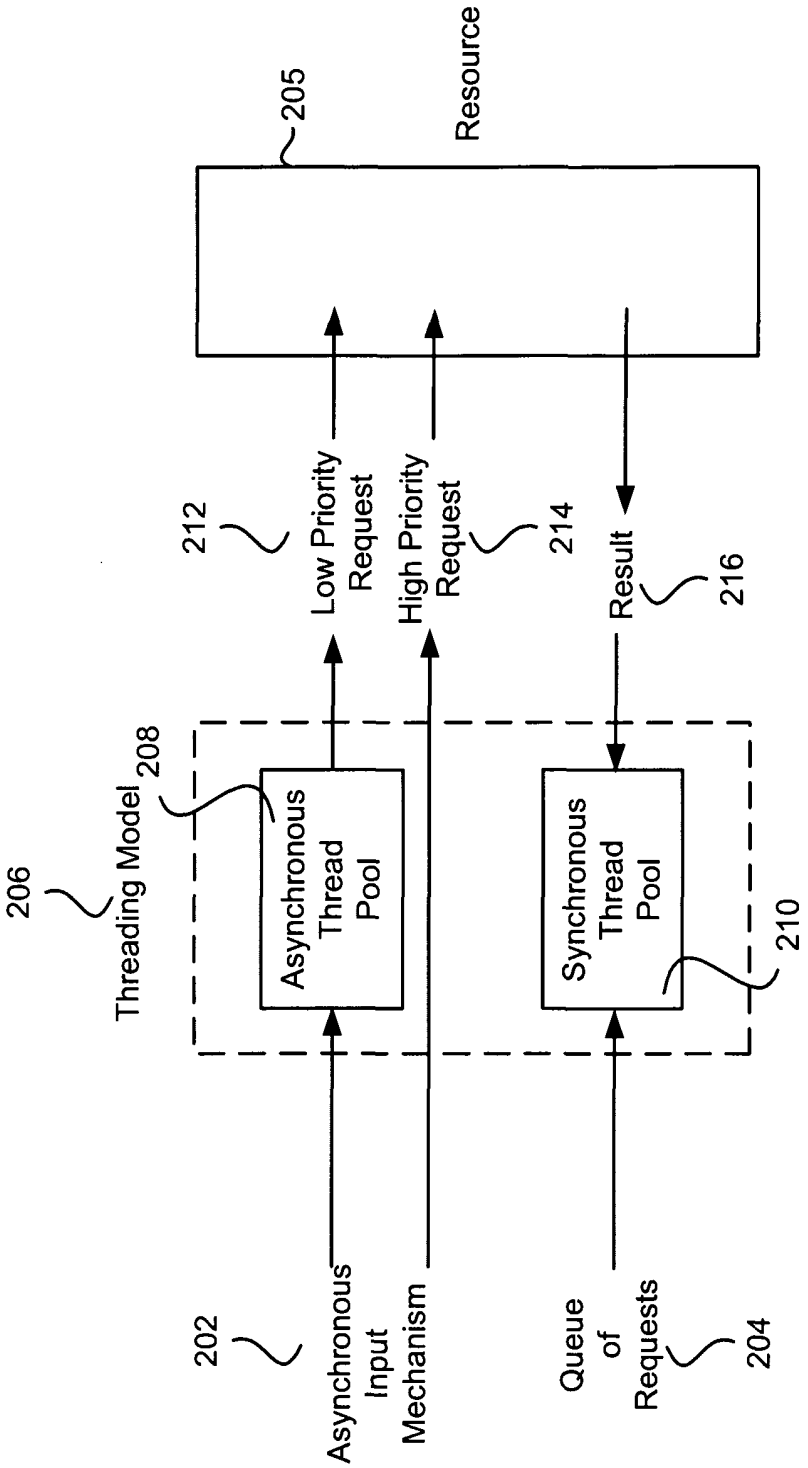

205
Resource
212
Low Priority Request
214
High Priority Request
216
Result
206
Threading Model
208
Asynchronous Thread Pool
Synchronous Thread Pool
210
202
Asynchronous Input Mechanism
Queue of Requests
204

FIGURE 2

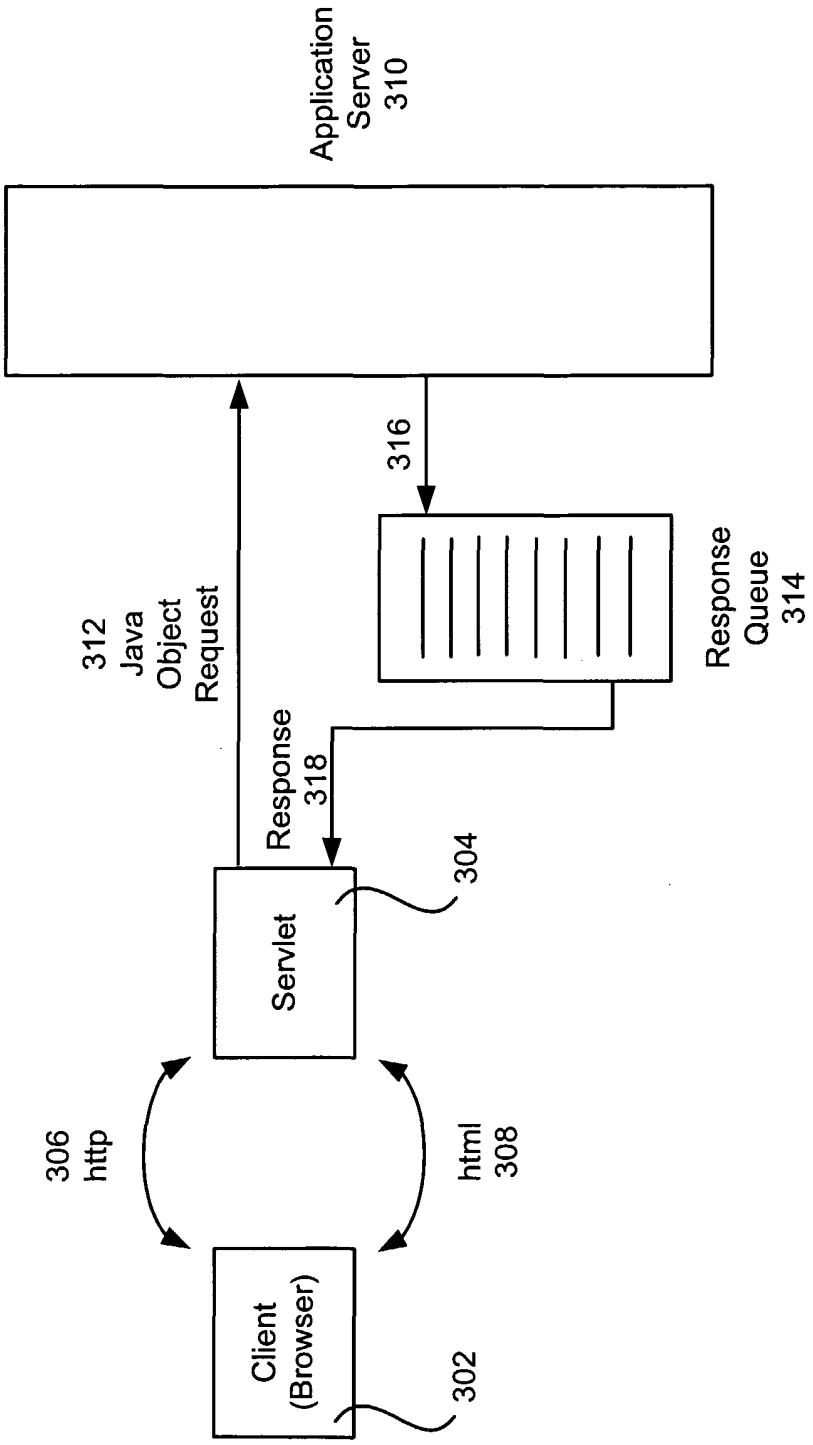
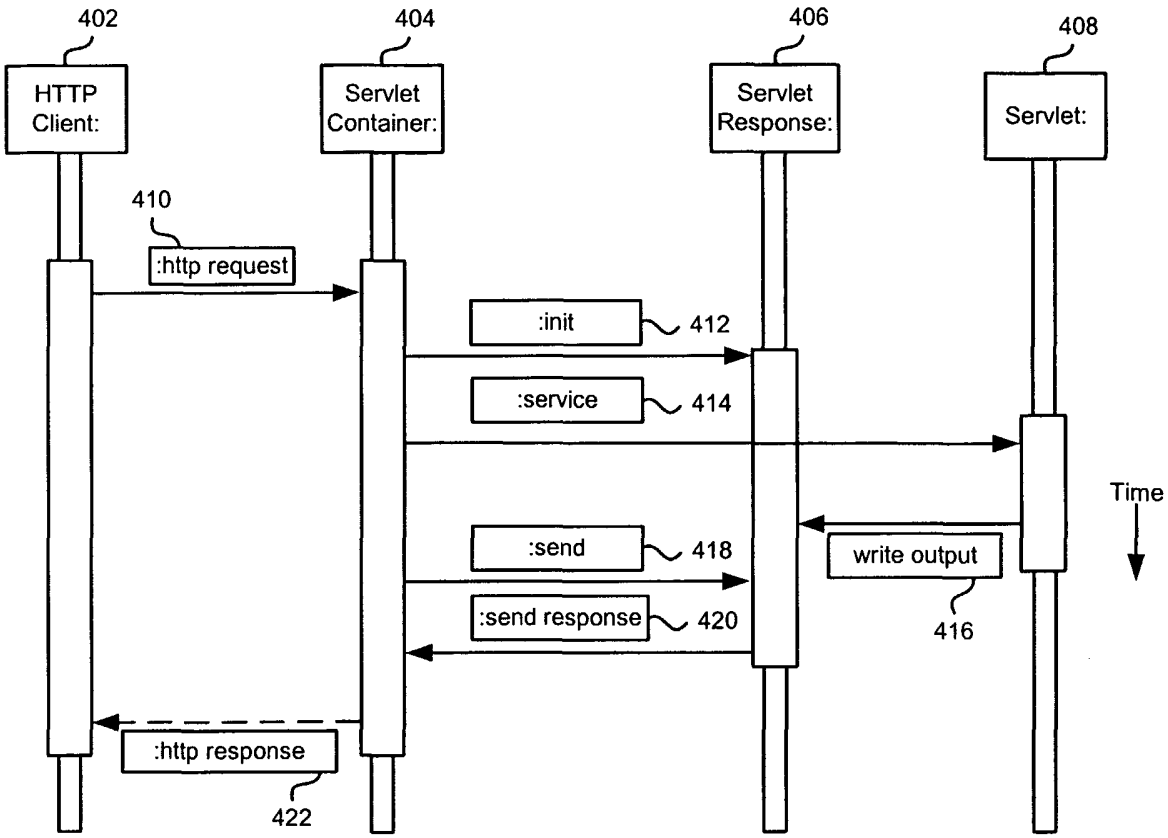
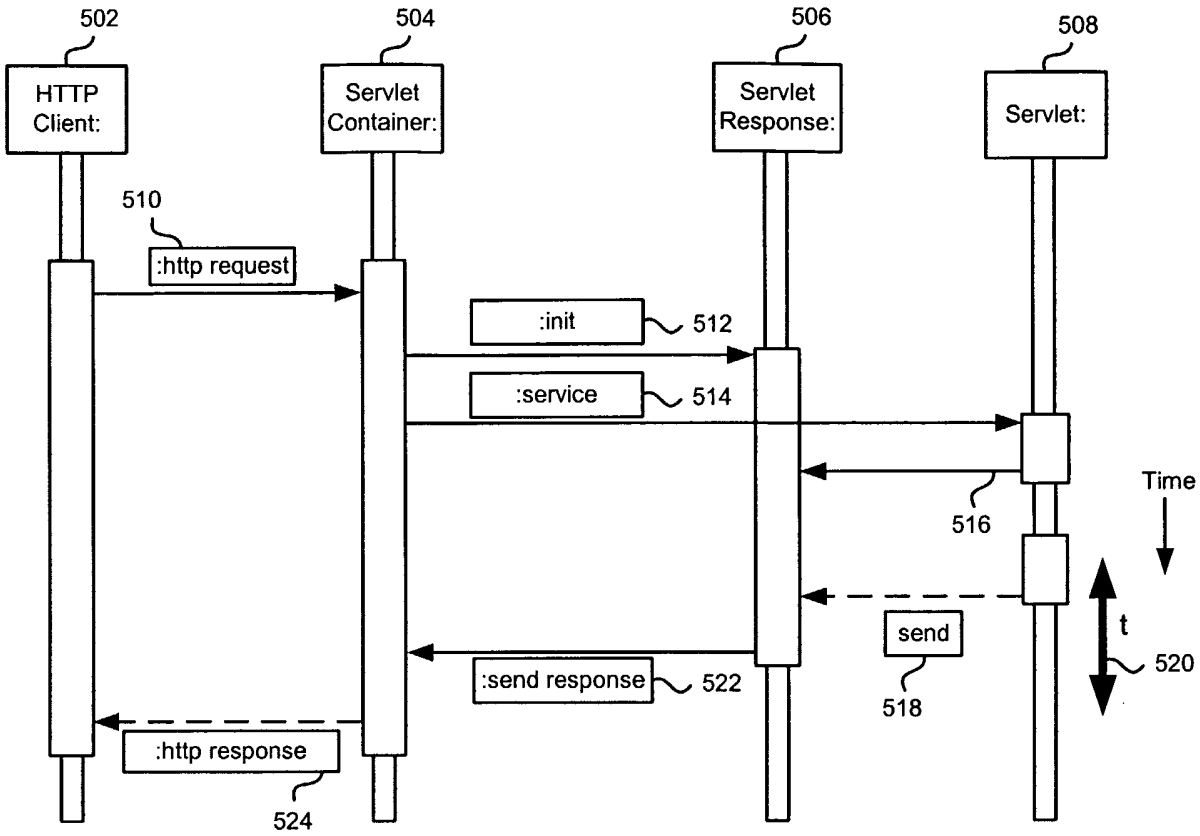

Application Server 310
316
Response Queue 314
312 Java Object Request
Response 318
304
Servlet
306 http
html 308
Client (Browser)
302

FIGURE 3



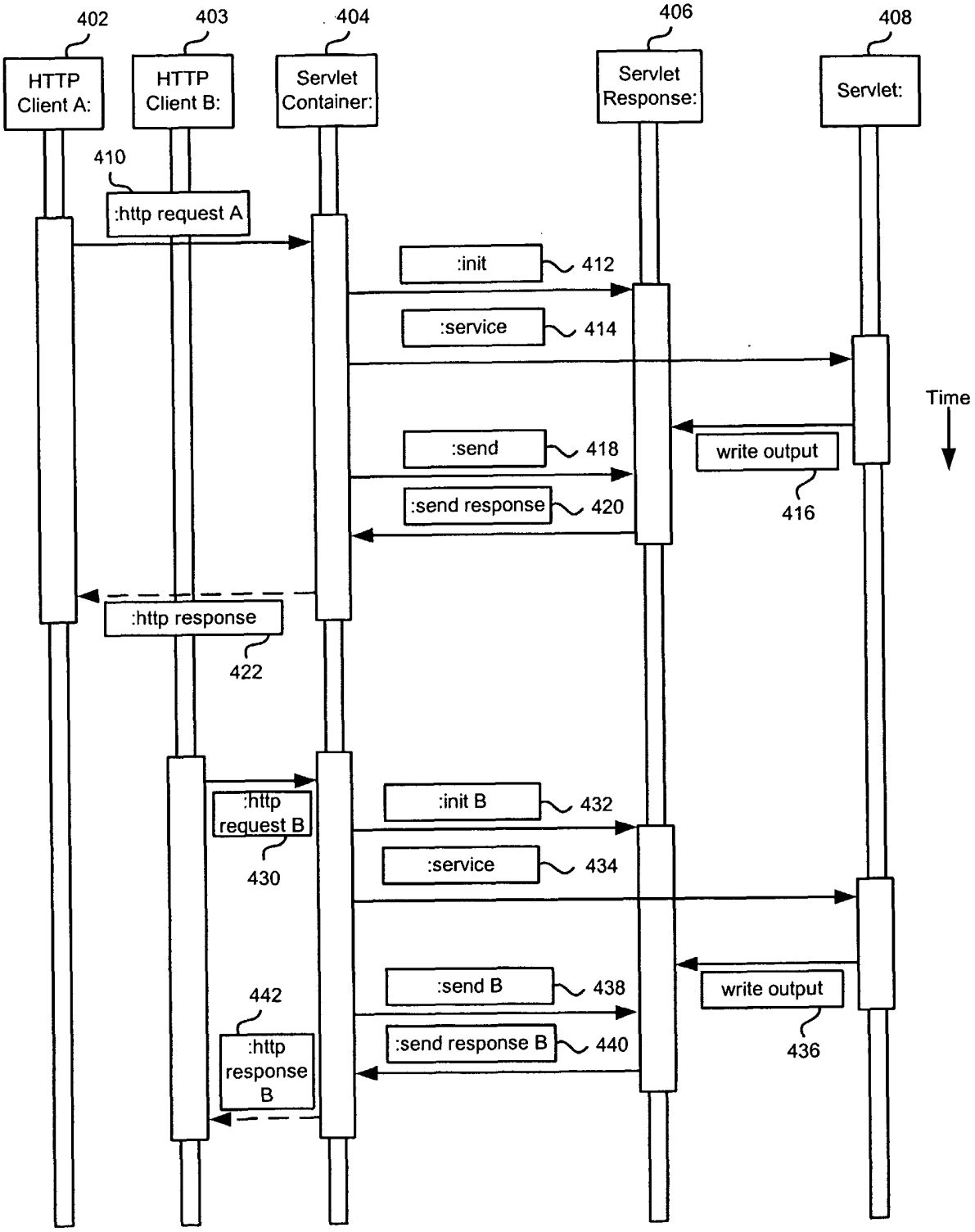
(Single Request)

FIGURE 4



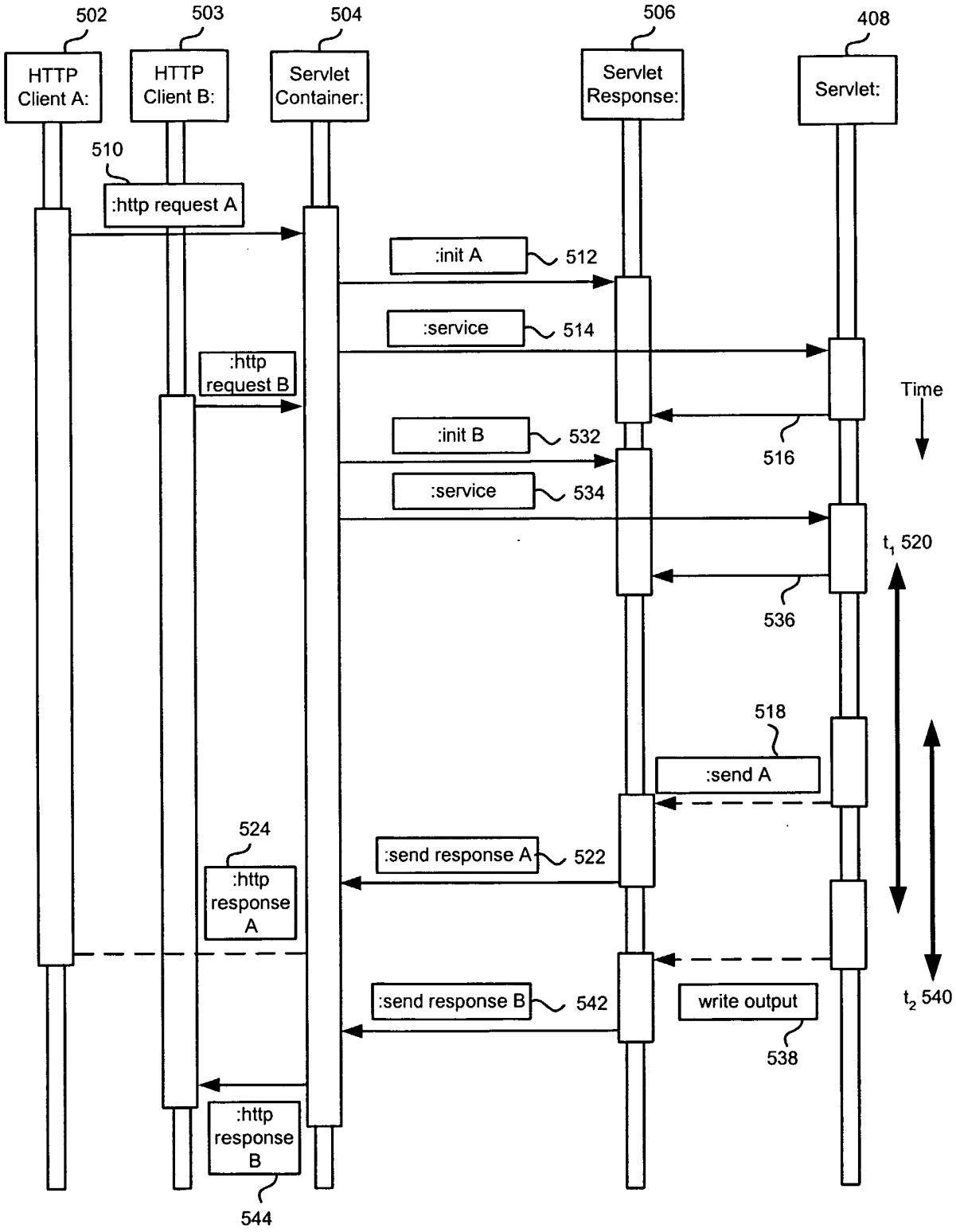
(Single Request)

FIGURE 5



(Multiple Requests)

FIGURE 6



(Multiple Requests)

FIGURE 7

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US02/31727

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) :G06F 15/16
US CL :709/203, 218, 219, 311; 707/511, 501, 513
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)

U.S. : 709/203, 218, 219, 311, 314; 707/511, 501, 513

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 practicable, search terms used)

Please See Extra Sheet.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 6,292,933 B1 (BAHRS ET AL.) 18 SEPTEMBER 2001, ABSTRACT, COL. 1, 17-67, COL. 2, LINES 1-67, COL. 3, LINES 1-67, COL. 4, LINES 1-67, COL. 5, LINES 1-7, COL. 14, LINES 23-67, COL. 15, LINES 1-65, COL. 16, LINES 56-67, COL. 17, LINES 1-60.	1-12
A	US 6,292,792 B1 (BAFFES ET AL.) 18 SPETEMBER 2001, ABSTRACT,	1-12
Y	US 5,987,454 A (HOBBS) 16 NOVEMBER 1999, ABSTRACT, COL. 7, LINES 34-67, COL. 8, LINES 1-67, COL. 9, LINES 1-12, COL. 10, LINES 30-67, COL. 11, LINES 1-17, COL. 12, LINES 40-67, COL. 13, LINES 65-67, COL. 14, LINES 1-65.	1-12

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents:	"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
"A"	document defining the general state of the art which is not considered to be of particular relevance	"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
"E"	earlier document published on or after the international filing date	"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
"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)	"&"	document member of the same patent family
"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		

Date of the actual completion of the international search
03 DECEMBER 2002

Date of mailing of the international search report
02 JAN 2003

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No. (703) 305-3230

Authorized officer
WILLIAM C. VAUGHN, JR.
Telephone No. (703) 305-9700

Form PCT/ISA/210 (second sheet) (July 1998)★

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US02/31727

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y, P	US 6,327,628 B1 (ANUFF ET AL.) 04 DECEMBER 2001, ABSTRACT, COL. 1, LINES 59-67, COL. 2, LINES 57-67, COL. 4, LINES 15-67, COL. 5, LINES 1-67, COL. 6, LOINES 1-65.	1-12
Y, P	US 6,480,865 B1 (LEE ET AL.) 12 NOVEMBER 2002, ABSTRACT, FIGURE 1, COL. 1, LINES 22-67, COL. 2, LINES 1-67, COL. 3, LINES 1-67, COL. 4, LINES 5-15, COL. 39-67.	1-12

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US02/31727

B. FIELDS SEARCHED
Electronic data bases consulted (Name of data base and where practicable terms used):

EAST, NPL
search terms: servlet, http, method, class, java server page, rmi, rpc, remote method invocation, remote procedure call, library, database, java database, queue, container, java messaging, object oriented