

(12) STANDARD PATENT
(19) AUSTRALIAN PATENT OFFICE

(11) Application No. **AU 2002362656 B9**

(54) Title
System for integrating java servlets with asynchronous messages

(51) International Patent Classification(s)
G06F 15/00 (2006.01) **H04L 29/06** (2006.01)
G06F 9/46 (2006.01) **H04L 29/08** (2006.01)
G06F 15/16 (2006.01)

(21) Application No: **2002362656** (22) Date of Filing: **2002.10.04**

(87) WIPO No: **WO03/032181**

(30) Priority Data

(31) Number	(32) Date	(33) Country
60/327,530	2001.10.05	US
10/264,973	2002.10.03	US

(43) Publication Date: **2003.04.22**

(43) Publication Journal Date: **2003.07.03**

(44) Accepted Journal Date: **2008.05.29**

(48) Corrigenda Journal Date: **2008.06.19**

(71) Applicant(s)
Bea Systems, Inc.

(72) Inventor(s)
Pullara, Sam;Brown, Dave;Messinger, Adam

(74) Agent / Attorney
Davies Collison Cave, 1 Nicholson Street, Melbourne, VIC, 3000

(56) Related Art
US 5987454
US 20010047385
US 6247044
US 6701438
US 6279030
WO 01/90945
US 6292933

Abstract:

In a traditional application server that uses servlets, when 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 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 (510). In one embodiment, the invention provides an extension to the Servlet API which allows the service method (514) to return (516), and thus the thread to be freed, before the response is ready to be sent. Then when the asynchronous event later (520) occurs the response may be completed and sent (518, 522, 524).

2002362656 12 May 2008

- 1 -

SYSTEM FOR INTEGRATING JAVA SERVLETS WITH ASYNCHRONOUS MESSAGES

Copyright Notice:

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

Claim of Priority:

This application claims priority from provisional application "SYSTEM FOR
15 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:

20 The invention relates to a system for messaging and/or communicating service
requests between an HTTP client and a server; and a method for messaging
and/or communicating service requests between an HTTP client and a server. For
example, the present invention generally relates to application and transaction
25 servers and particularly to a system for supporting message queuing and threads
with multiple execute queues.

Cross References:

30 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

- 2 -

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.

5

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 faster 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.

15

[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

20

25

- 3 -

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

- 4 -

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

- 5 -

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.

5 **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

- 6 -

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

2002362656 12 May 2008

- 7 -

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

One problem with current Servlet APIs is the completely synchronous
5 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
10 event must occur before the response can be sent, because the thread running
the servlet must block until the event occurs.

It is generally desirable to overcome or ameliorate one or more of the above
described difficulties, or to at least provide a useful alternative.

15 **Summary of the Invention:**

In accordance with one aspect of the present invention, there is provided a system
for messaging and/or communicating service requests between an HTTP client
and a server, comprising:

20 a server that includes a servlet container process, a servlet process, and a
servlet response handler process executing on the server;
an HTTP software interface provided by the servlet container that receives
requests from an HTTP client, wherein a request from the HTTP client is
transmitted by the servlet container to the servlet; and
25 wherein the servlet upon receiving the request from the servlet container
immediately sets a response code to be used with a subsequent response,
releases the response handler to interleave the processing of subsequent
requests from the HTTP client or from other HTTP clients, and then
processes the original request by locating the required information, resetting
30 the response code, and sending the information to the HTTP client using the
servlet response handler.

In accordance with another aspect of the present invention, there is provided a method for messaging and/or communicating service requests between an HTTP client and a server, comprising the steps of:

5 receiving at a server that includes a servlet container process, a servlet process, a servlet response handler process and an HTTP software interface executing thereon, a request from an HTTP client to access the servlet within the servlet container; and immediately
10 setting a response code to be used with a subsequent response, releasing the servlet response handler to interleave the processing of subsequent requests from the HTTP client or from other HTTP clients, processing the original request by locating the required information, resetting the response code, and sending the information to the HTTP client using the servlet response handler.

15 The invention preferably 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.

20

In accordance with a preferred embodiment of 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
25 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.

30

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.

5

Brief Description of the Drawings:

Preferred embodiments of the present invention are hereafter described, by way of non-limiting example only, with reference to the accompanying drawings, in which:

10 Figure 1 shows an illustration of a J2EE compatible architecture that can utilize the present invention.

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

Figure 3 shows a diagram of a synchronous threading process.

15 Figure 4 shows a lifecycle of a single HTTP request that is processed using traditional methods.

Figure 5 shows a lifecycle of a single HTTP request processed using asynchronous messaging.

20 Figure 6 shows a lifecycle of a plurality of HTTP requests processed using traditional methods.

Figure 7 shows a lifecycle of a plurality of HTTP requests processed using asynchronous messaging.

25

Detailed Description of Preferred Embodiments of the Invention:

Broadly described, the invention preferably 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.

30

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

2002362656 12 May 2008

- 10 -

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.

5

In one preferred 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

10 this mechanism is the use of JMS in conjunction with servlets.

In this preferred 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
15 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.

20

The invention is preferably 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
25 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
30 requests while tolerating user code that performs blocking operations.

2002362656 12 May 2008

- 10a -

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
5 callbacks usually queue the request for later processing by the synchronous thread pool. However certain non-blocking, priority requests are serviced 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
10 block there are usually a low number of them, perhaps one per processor (CPU).

- 11 -

[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.

- 12 -

[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**

- 13 -

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 **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

- 14 -

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.

5 **[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.

10

15

[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

20

25

- 15 -

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**.
5 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
10 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

15 **[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.

20 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
25 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.

- 16 -

5 **[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.

10 **[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.

20

Servlets

25 **[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

- 17 -

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

2002362656 12 May 2008

- 18 -

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
5 defined by the following claims and their equivalence.

Throughout this specification and the claims which follow, unless the context requires otherwise, the word "comprise", and variations such as "comprises" and "comprising", will be understood to imply the inclusion of a stated integer or step or
10 group of integers or steps but not the exclusion of any other integer or step or group of integers or steps.

The reference in this specification to any prior publication (or information derived from it), or to any matter which is known, is not, and should not be taken as an
15 acknowledgment or admission or any form of suggestion that that prior publication (or information derived from it) or known matter forms part of the common general knowledge in the field of endeavour to which this specification relates.

2002362656 12 May 2008

- 19 -

THE CLAIMS DEFINING THE INVENTION ARE AS FOLLOWS:

1. A system for messaging and/or communicating service requests between an HTTP client and a server, comprising:
 - 5 a server that includes a servlet container process, a servlet process, and a servlet response handler process executing on the server;
 - an HTTP software interface provided by the servlet container that receives requests from an HTTP client, wherein a request from the HTTP client is transmitted by the servlet container to the servlet; and
 - 10 wherein the servlet upon receiving the request from the servlet container immediately sets a response code to be used with a subsequent response, releases the response handler to interleave the processing of subsequent requests from the HTTP client or from other HTTP clients, and then processes the original request by locating the required information, resetting
 - 15 the response code, and sending the information to the HTTP client using the servlet response handler.
2. The system of claim 1 wherein the subsequent response is triggered by a response command from the HTTP client to send additional response data.
- 20 3. The system of claim 1 wherein the system includes an extension to the HTTP interface that allows a service method to return, and the thread handling that service method to be released before the response is ready to be sent.
- 25 4. The system of claim 1 further including a Java Messaging Service (JMS), wherein the servlet builds a portion of a response, queues a JMS message requesting the data, and creates a response object as the response code, which is later retrieved when another JMS message containing the
- 30 requested data arrives.

2002362656 12 May 2008

- 20 -

5. The system of claim 1 wherein the system sets a period of time for which to send the subsequent response, after which period of time the servlet sends a send signal to the response handler, which then signals the servlet container to send the subsequent response to the client.
- 5 6. The system of claim 1 wherein the system is used to provide information within a search function to a user of a web page, and wherein the system provides partial items of information to the user's screen, while other items of information are returned as subsequent responses to the user's screen as the servlet finds the appropriate data.
- 10 7. A method for messaging and/or communicating service requests between an HTTP client and a server, comprising the steps of:
- 15 receiving at a server that includes a servlet container process, a servlet process, a servlet response handler process and an HTTP software interface executing thereon, a request from an HTTP client to access the servlet within the servlet container; and immediately
- 20 setting a response code to be used with a subsequent response, releasing the servlet response handler to interleave the processing of subsequent requests from the HTTP client or from other HTTP clients, processing the original request by locating the required information, resetting the response code, and sending the information to the HTTP client using the servlet response handler.
- 25 8. The method of claim 7 wherein the subsequent response is triggered by a response command from the HTTP client to send additional response data.
- 30 9. The method of claim 7 wherein the system includes an extension to the HTTP interface that allows a service method to return, and the thread handling that service method to be released before the response is ready to be sent.

2002362656 12 May 2008

- 21 -

10. The method of claim 7 further including a Java Messaging Service (JMS), wherein the servlet builds a portion of a response, queues a JMS message requesting the data, and creates a response object as the response code, which is later retrieved when another JMS message containing the requested data arrives.
11. The method of claim 7 wherein the system sets a period of time for which to send the subsequent response, after which period of time the servlet sends a send signal to the response handler, which then signals the servlet container to send the subsequent response to the client.
12. The method of claim 7 wherein the system is used to provide information within a search function to a user of a web page, and wherein the system provides partial items of information to the user's screen, while other items of information are returned as subsequent responses to the user's screen as the servlet finds the appropriate data.
13. A system for messaging and/or communicating service requests between an HTTP client and a server substantially as hereinbefore described, with reference to the accompanying drawings.
14. A method for messaging and/or communicating service requests between an HTTP client and a server substantially as hereinbefore described, with reference to the accompanying drawings.

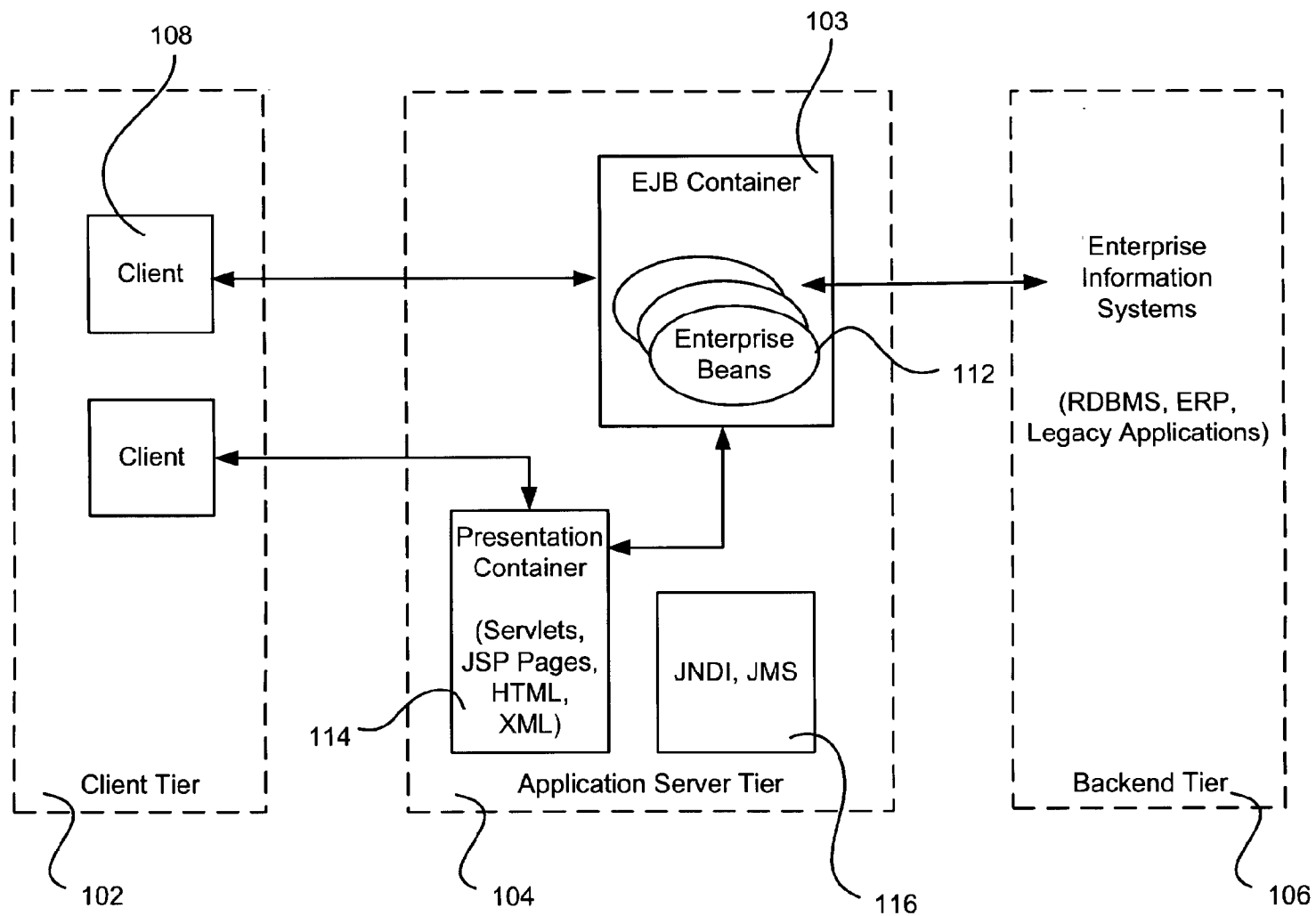


FIGURE 1

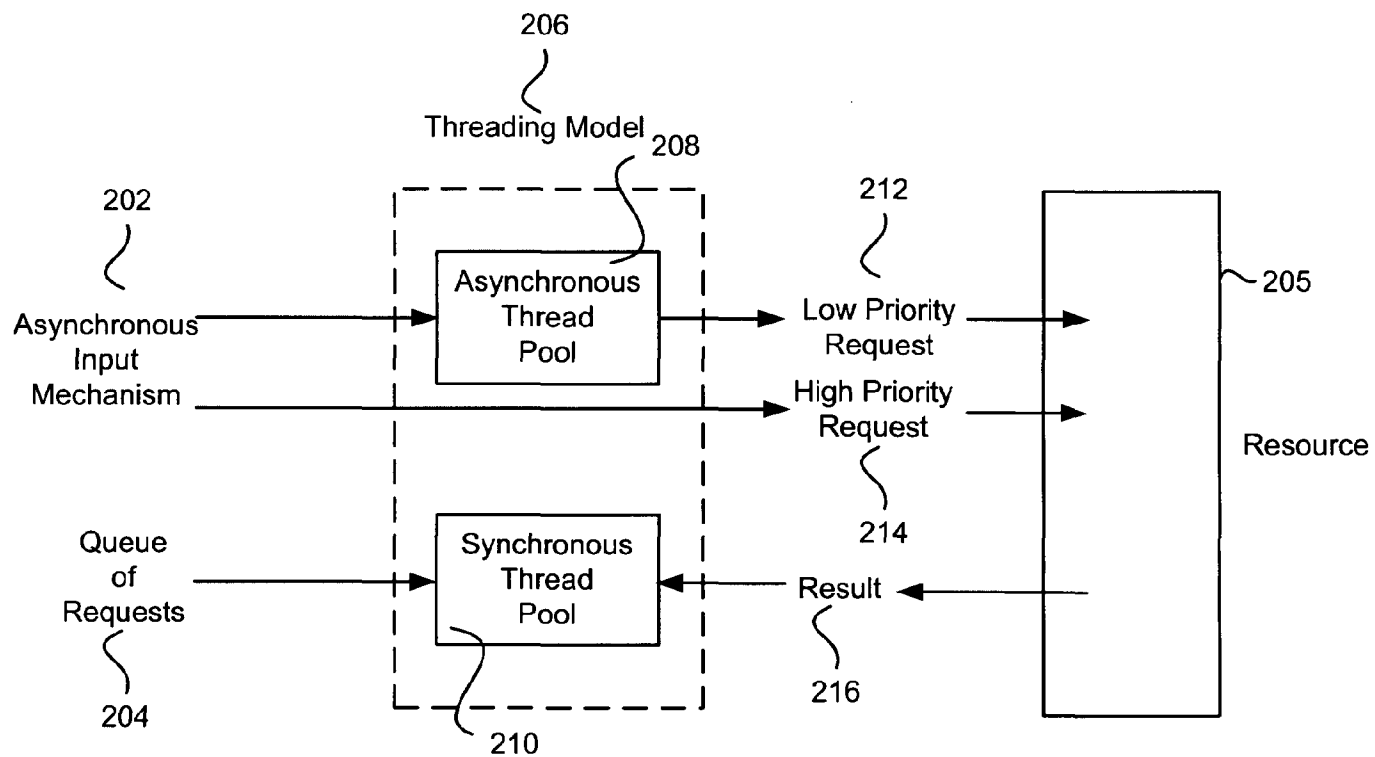


FIGURE 2

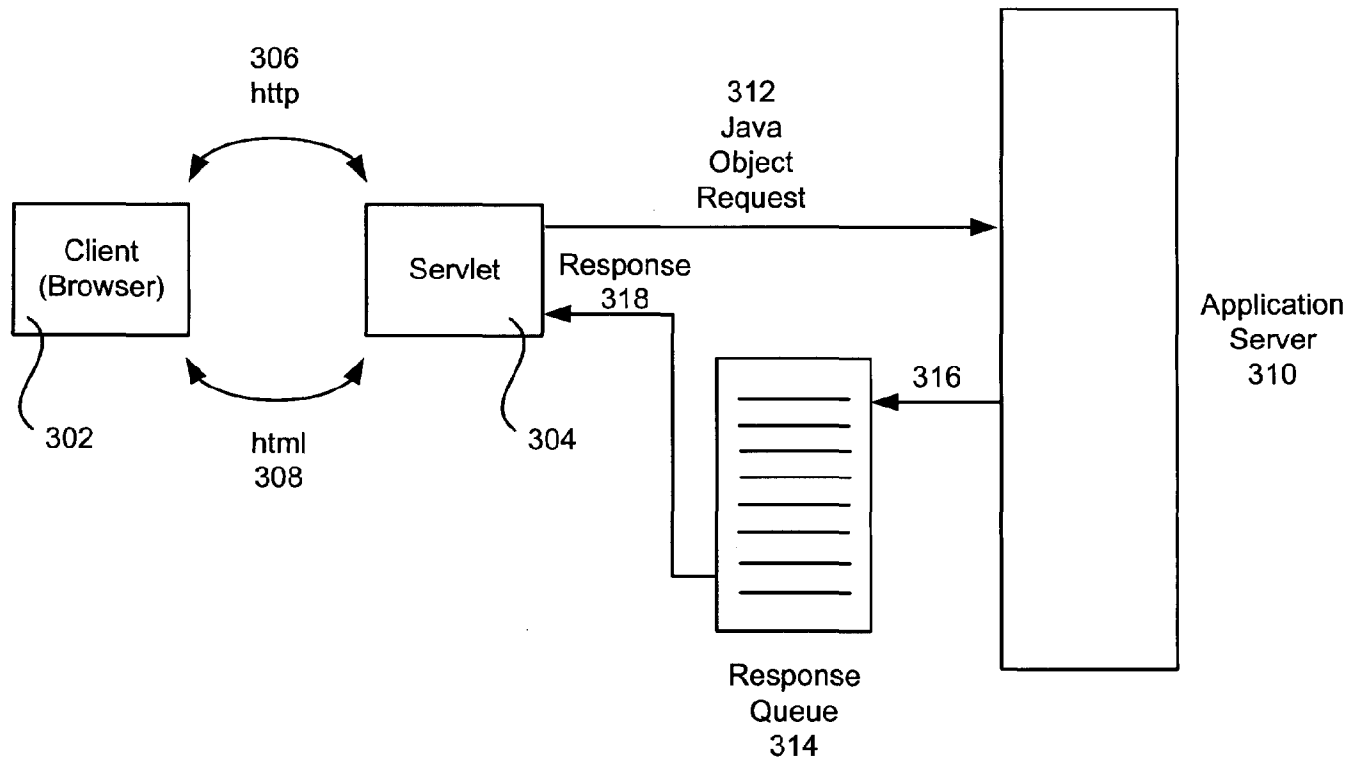
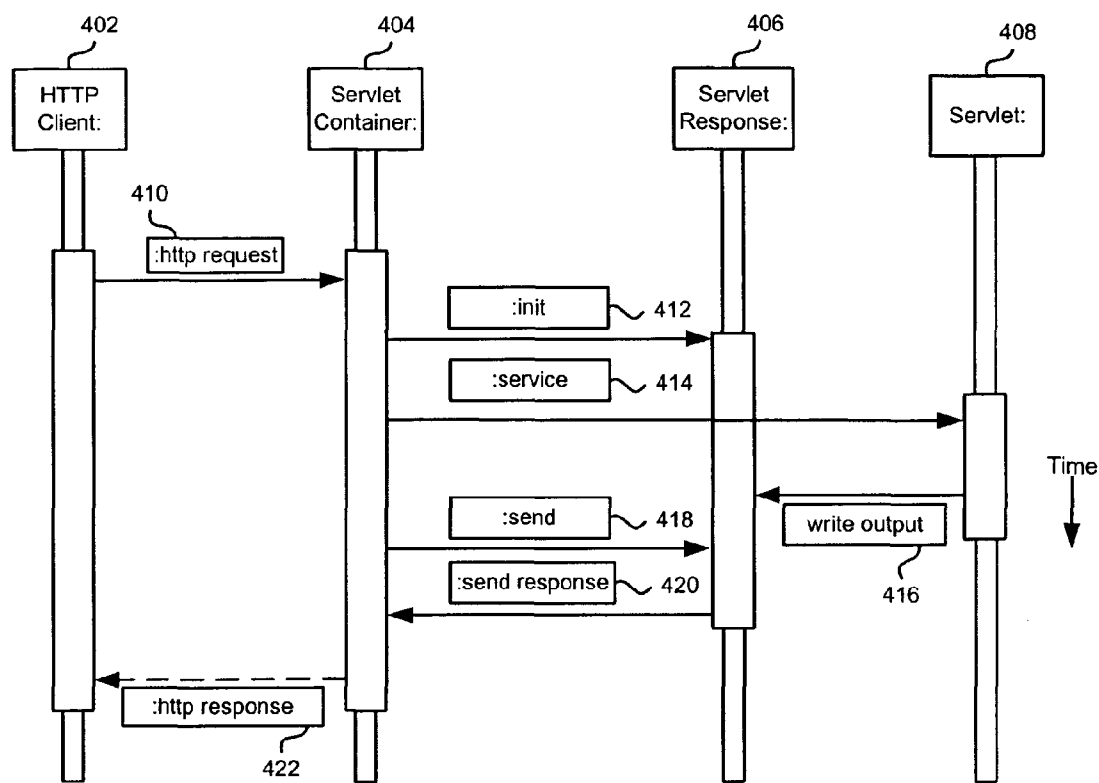
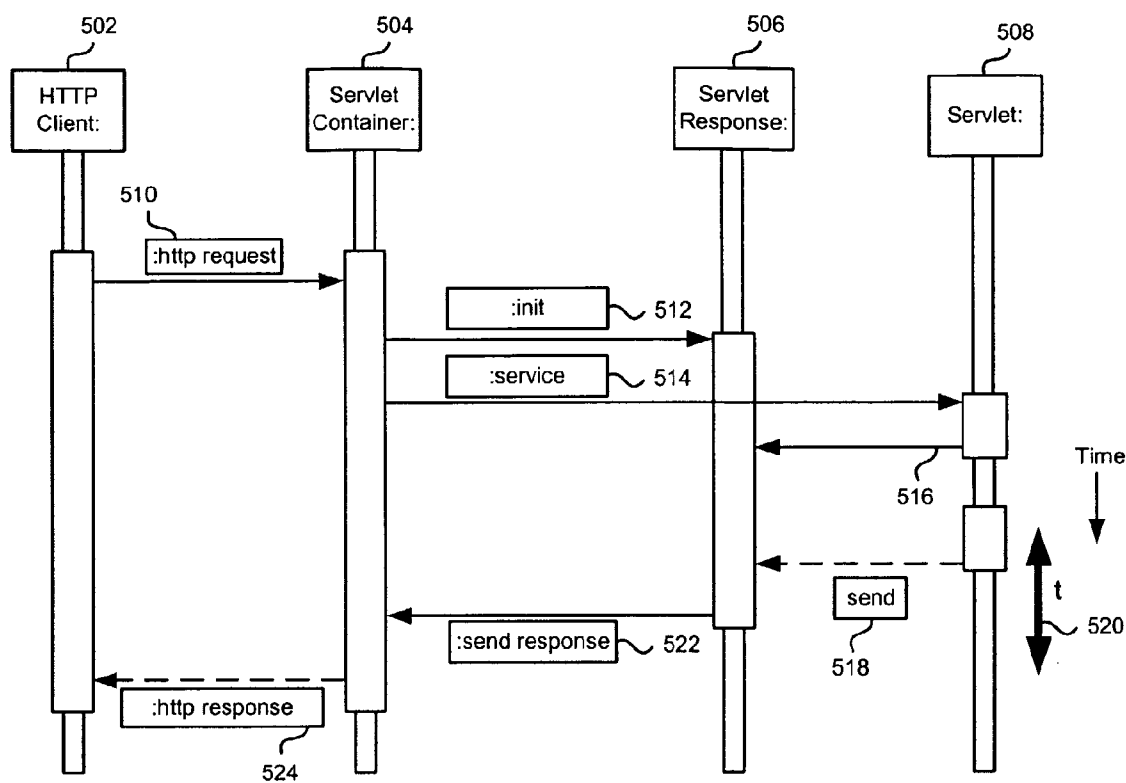


FIGURE 3



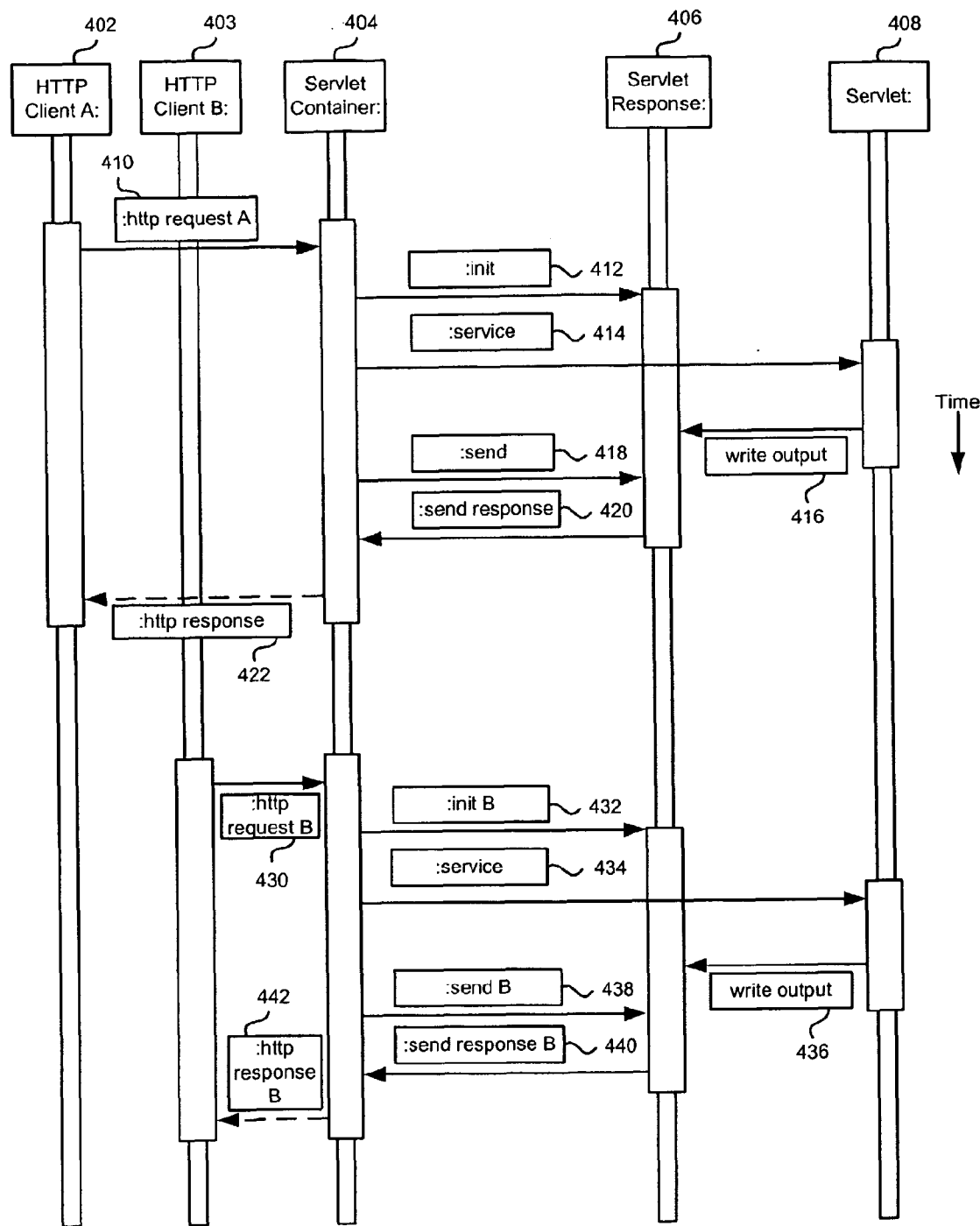
(Single Request)

FIGURE 4

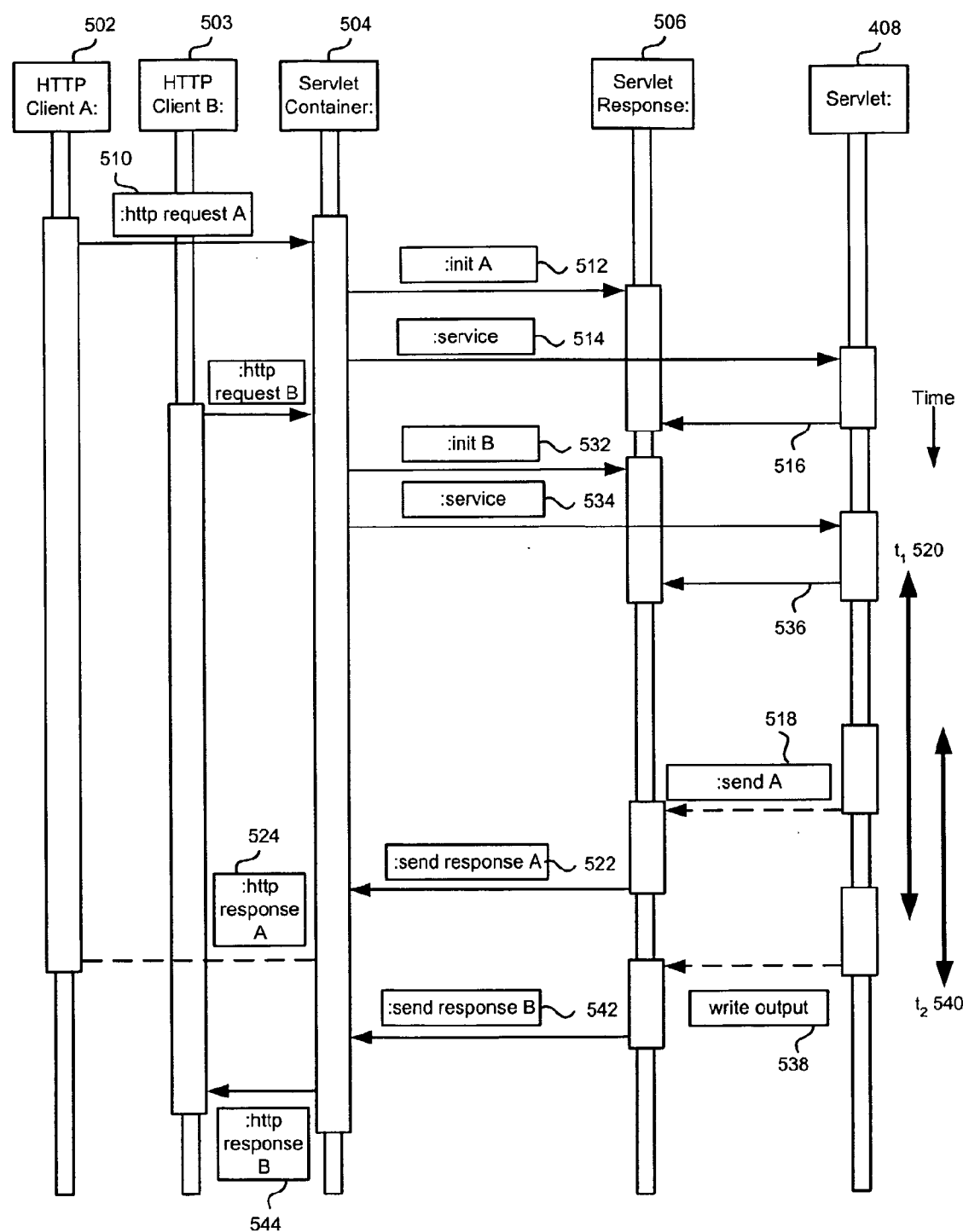


(Single Request)

FIGURE 5



(Multiple Requests)
FIGURE 6



(Multiple Requests)
FIGURE 7