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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
13 November 2003 (13.11.2003)

PCT

(10) International Publication Number
WO 03/094019 A1

(51) International Patent Classification[7]: G06F 15/16, 13/14

(21) International Application Number: PCT/US03/13621

(22) International Filing Date: 1 May 2003 (01.05.2003)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/377,332 2 May 2002 (02.05.2002) US
10/293,059 13 November 2002 (13.11.2002) US

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

(72) Inventors: **POTTER, Timothy**; 4900 S. Ulster Street, #8-106, Denver, CO 80237 (US). **UPTON, Mitch**; 10099 Briargrove Way, Highlands Ranch, CO 80126 (US). **GOLDING, Christa**; 407 W. English Sparrow Trail, Littleton, CO 80129 (US).

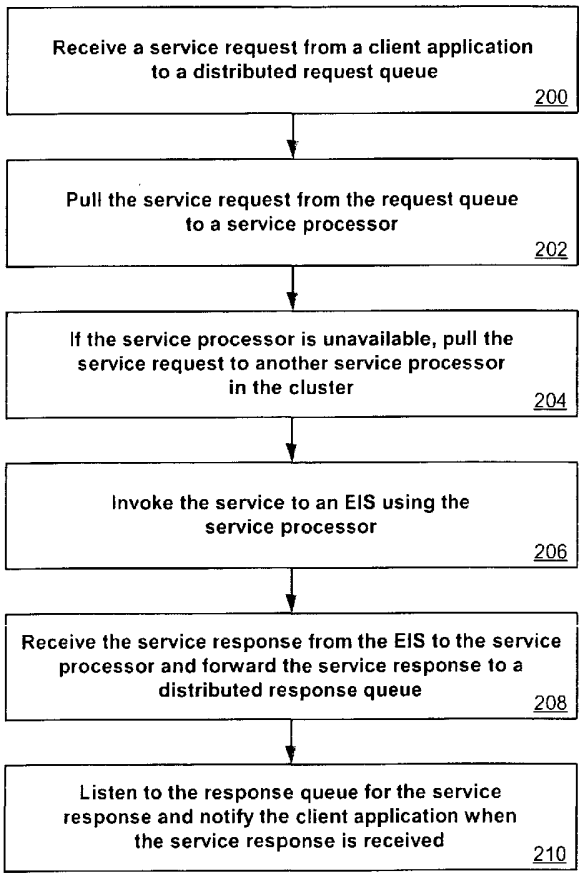
(74) Agent: **MEYER, Sheldon R.**; Fliesler Dubb Meyer & Lovejoy LLP, Four Embarcadero Center, Suite 400, San Francisco, CA 94111-4156 (US).

(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, NI, NO, NZ, OM, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VC, 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,

[Continued on next page]

(54) Title: HIGH AVAILABILITY FOR ASYNCHRONOUS REQUESTS



(57) Abstract: Highly-available processing of an asynchronous request can be accomplished in a single transaction. A distributed request queue receives a servuce request from a client application (200). A service processor is deployed on each node of a cluster containing the distributed request queue. A service processor pulls the service request from the request queue and invokes the service for the request (202). If that service processor fails, another service processor in the cluster can service the request (204). The service processor receives a service response from the invoked service (206) and forwards the service response to a distributed response queue (208). The distributed response queue holds the service response queue holds the service response until the response is retrieved for the client application (210).

WO 03/094019 A1

ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

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.

Published:

— *with international search report*

HIGH AVAILABILITY FOR ASYNCHRONOUS REQUESTS

COPYRIGHT NOTICE

CLAIM OF PRIORITY

This application claims priority to U.S. Provisional Patent Application No. 60/377,332, filed May 2, 2002, entitled "HIGH AVAILABILITY FOR ASYNCHRONOUS REQUESTS," which is hereby incorporated herein by reference.

CROSS-REFERENCED CASES

The following applications are cross-referenced and incorporated herein by reference:

U.S. Patent Application No. 10/271,194 entitled "Application View Component for System Integration," by Mitch Upton, filed October 15, 2002.

U.S. Patent Application No. 10/293,674 entitled "High Availability Event Topic," by Tim Potter et al., filed November 13, 2002.

U.S. Patent Application No. 10/293,655 entitled "High Availability Application View Deployment," by Tim Potter et al., filed November 13, 2002.

U.S. Patent Application No. 10/293,656 entitled "High Availability for Event Forwarding," by Tim Potter et al., filed November 13, 2002.

FIELD OF THE INVENTION

The present invention relates to the availability of services such as JMS across a network or in a server cluster.

BACKGROUND

In present application integration (AI) systems, there can be several single points of failure. These single points of failure can include deployment or management facilities, event forwarding, event topics, remote clients, event subscriptions, response listeners, and response queues. Each of these features is tied to a single server within a server cluster. If that single server crashes, the entire AI application can become irreparably damaged and must be rebooted via a server reboot.

Single points of failure such as request and response queue are used for processing asynchronous requests. Current implementations of asynchronous service request processing utilize a single physical request queue and response queue per server instance. In the event of a node failure, all asynchronous requests and responses within a given JMS server, for example, become unavailable until the JMS server is restarted.

BRIEF SUMMARY

Systems and methods in accordance with the present invention can overcome deficiencies in prior art systems by allowing for high-availability processing of asynchronous requests in a single transaction. A distributed request queue can be used to receive and

store a service request, such as from a user or client application. A service processor can pull the service request from the request queue and invoke the service for the service request, such as to an enterprise information system. The service processor can receive the service response from the invoked service and forward the service response to a distributed response queue. The distributed response queue can hold the service response until the response is retrieved for the user or client application. An application view client can act on behalf of the user or client application, sending the service request to the distributed request queue and retrieving the service response from the distributed response queue. The application view client can generate failure recovery semantics for the client application in the event of a failure. The application view can also determine whether any service responses are waiting in the distributed response queue for the client application.

These systems and methods can be used in a server cluster. There can be a service processor deployed on every node in the cluster, each of which can listen to a given distributed request queue. This allows a service to be migrated between nodes in the cluster in the event of a node failure.

Other features, aspects, and objects of the invention can be obtained from a review of the specification, the figures, and the claims.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a diagram of a system in accordance with one embodiment of the present invention.

Figure 2 is flowchart for a method that can be used with the system of Figure 1.

DETAILED DESCRIPTION

A system and method in accordance with one embodiment of the present invention can overcome deficiencies in present asynchronous messaging systems by taking advantage of asynchronous request and response queues, as well as asynchronous request and response processors. A client may wish to invoke a service asynchronously in order to begin and/or continue processing other matters, instead of simply waiting for the response. For example, a long running process such as a batch process run against an SAP system or database can take minutes or even hours. Asynchronous requests can allow a client to send the request and then move on to other business.

The use of server clustering allows an AI component to be used in a scalable and highly available fashion. A highly available component does not have any single points of failure, and can have the ability to migrate services from failed nodes to live nodes in a cluster. Any service offered by the AI component can be targeted to several nodes in a cluster. In the event of a node failure in the cluster, the services located on the failed node can be migrated to another live node in the cluster.

In the event of a crash of a cluster or managed server, the AI application can continue accepting new work. The acceptance of new work can include deploying new and undeploying old application views and connection factories, monitoring of old application views and connection factories, delivering events from adapters, and servicing both synchronous and asynchronoushronous service invocations. An AI application can also support the manual migration of services on the failed node to a live node, such as a singleton message-driven Enterprise JavaBean (MDB) listening on a physical destination managed by a failed JMS server. Application integration

can use a singleton MDB if a customer needs ordered event processing, for example. An AI application can notify users in an understandable and/or predictable way that in-flight transactions have been cancelled or rolled-back, and should be retried. Wherever possible, an AI application can retry the transaction after reestablishing connections to make use of resources on another live server.

In the event of an administration (admin) server failure, an AI application can do all the tasks mentioned with respect to a crash of a cluster or managed server. The AI application can also notify users that deployment or undeployment is unavailable while the admin server is unavailable. The AI application can still boot or reboot successfully using the previous domain and/or server configuration.

A system and method in accordance with one embodiment of the present invention allows asynchronous requests and responses to be available within a given JMS server, even in the event of a node failure. Request and response queues, such as ASYNC_REQUEST_QUEUE and ASYNC_RESPONSE_QUEUE, can be deployed as distributed queues in a cluster. A request processor, such as AsyncServiceRequestProcessor, can be packaged as an MDB. Such a system allows the processing of asynchronous requests and responses even if the JMS server that accepted the requests crashes or becomes otherwise unavailable.

In the event that a physical queue fails before an asynchronous service request is received by the appropriate MDB, the request can be unavailable until the physical queue comes back on line. This can hold true for asynchronous service responses. Using a system in accordance with one embodiment of the present invention, an asynchronous service processor MDB can be deployed

on a single distributed JMS queue, such as ASYNC_REQUEST_QUEUE. This deployment removes the need to maintain and manage a pool of asynchronous request processor threads. An asynchronous service processor MDB can be last in the deployment order for the AI application, and can be deployed from a JAR file such as "ai-asyncprocessor-ejb.jar."

Figure 1 shows an example of a high-availability asynchronous service processing system in accordance with one embodiment of the present invention. An application view client **100** has the ability to generate and deal with failure recovery semantics without the user having any knowledge or input. For instance, a client application that sends off a request might crash or otherwise become unavailable at some point before the response is received. When the response is ready to be returned, the response can sit in an asynchronous response queue **112** until the client comes back. When the client **100** is available again, the client will want to receive the response. Since the system is utilizing distributed queues, the client application would need to go out to the server and determine whether there are any responses from previous requests that were sent before the failure. The application view client **100** can take care of this determination behind the scenes, such that the user or client application does not need to do anything to find the response.

The user or client application making the request can register a message listener **106**, such that the user or client application can be informed that a message is ready and waiting to be received. An asynchronous service processor **110** can pull a request off the asynchronous request queue **108**, invoke the asynchronous service against an Enterprise Information System (EIS) **118**, and wait for the response. When the asynchronous service response comes back, the asynchronous service processor **110** can put the response onto

the response queue **112**. In this embodiment, this processing is accomplished as a single transaction.

The application view client **100** can instantiate an application view instance **102**. The client **100** can have the option of supplying a durable client identifier at the time of construction. The durable client identifier can be used as a correlation identifier for asynchronous response messages. The client **100** can invoke an asynchronous service method, such as "invokeServiceAsync", and can pass a request document and response listener **104**, such as AsyncServiceResponseListener, to handle the response.

An application view instance **102** can create a service request object, such as AsyncServiceRequest, and can send the object to a request queue **108**, such as ASYNC_REQUEST_QUEUE. The service request object can contain the name of the destination to which the response listener is pinned. A service processor MDB **110** can use this information to determine the physical destination to receive the response. If the request object does not contain the name of a response destination, the service processor MBD **110** can use the destination set on the JMS message via a call to a method such as JMSReplyTo(). If a client only supplies a service response listener **104** to the application view, such as:

```
invokeServiceAsync(String serviceName, IDocument
    request, AsyncServiceResponseListener listener);
```

the application view can establish a JMS queue receiver to the JMS queue bound at a JNDI location provided by an application view Enterprise JavaBean (EJB) method, such as getAsyncResponseQueueJNDIName(). The application view instance **102** can use QueueReceiver::getQueue() to set the ReplyTo destination on the request message.

In a cluster, an asynchronous request queue **108** can be deployed as a distributed JMS queue. Each message can be sent to a single physical queue, and not be forwarded or replicated in any way. As such, the message is only available from the physical queue to which it was sent. If that physical queue becomes unavailable before a given message is received, the message or AsyncServiceRequest can be unavailable until that physical queue comes back on-line. It is not enough to send a message to a distributed queue and expect the message to be received by a receiver of that distributed queue. Since the message is sent to only one physical queue, there must be a queue receiver receiving or listening on that physical queue. Thus, an AI asynchronous service processor MDB can be deployed on all nodes in a cluster.

An asynchronous service processor MDB can receive the message from the queue in a first-in, first-out (FIFO) manner. The service processor can use the asynchronous service request object in a JMS ObjectMessage to determine the qualified name, service name, request document, and response destination of the application view. The asynchronous service processor **110** can use an application view EJB **114** to invoke the service synchronously. The service can be translated into a synchronous CCI-based request and/or response to the resource adapter **116**.

When an asynchronous service processor MDB **110** receives the response, the response can be encapsulated into an asynchronous service response object and sent to the response destination provided in the asynchronous service request object. The asynchronous service processor MDB **110** cannot just send the response to the asynchronous response queue **112**, the response needs to be sent to a specific physical destination. This specific physical destination, or queue, can have been established by the

application view instance **102** running on the client when, for example, an application view EJB method such as getAsyncResponseQueueJNDIName() was called.

If the client application fails and a new application view is created with the same durable client identifier, there is a chance that the new application view will be pinned to a different physical JMS queue than the JMS queue that the client was using prior to the failure. Consequently, the application view can use recover logic to query the other members for responses that match the durable client identifier once the client application restarts.

An application view message listener **106** instance, created when the application view instance **102** was instantiated, can receive the asynchronous service response message as a JMS ObjectMessage, and can pass the message to the asynchronous service response listener **104** supplied in the "invokeServiceAsync" call.

Figure 2 shows the steps of a method that can be used with the system of Figure 1. First, a service request is received to a distributed request queue from a client application **200**. The service request is pulled from the request queue to a service processor **202**. If the service processor is down, another service processor in the cluster pulls the service request **204**. A service is invoked for the service request, such as to an EIS **206**. The service response is retrieved by the service processor and forwarded to a distributed response queue for storage until retrieval from a client application **208**. A response listener listens to the response queue and notifies the client application when the service response is received **210**.

The foregoing description of preferred embodiments 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. Many modifications and variations will be apparent to one of ordinary skill 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.

What is claimed is:

1. A system for high-availability processing of asynchronous requests in a single transaction, comprising:

a distributed request queue for receiving and storing a service request;

a service processor for pulling the service request from the request queue and invoking the service for the service request, the service processor further receiving a service response for the service request from the invoked service; and

a distributed response queue for receiving the service response from the service processor and storing the service response.

2. A system according to claim 1, further comprising:

an enterprise information system containing the service invoked by the service processor.

3. A system according to claim 1, wherein:

said service processor is packaged as a message-driven Enterprise JavaBean.

4. A system according to claim 1, further comprising:

an application view client for sending the service request to the distributed request queue and retrieving the service response from the distributed response queue on behalf of a client application.

5. A system according to claim 4, wherein:

said application view client can generate failure recovery semantics for the client application.

6. A system according to claim 4, wherein:

the distributed response queue is adapted to store the service response until the response is retrieved by the application view client.

7. A system according to claim 4, wherein:

said application view is adapted to determine whether any service responses are waiting in the distributed response queue for the client application.

8. A system according to claim 4, further comprising:

a client identifier for identifying the client application, the client identifier used to process the service request and service response for the client application.

9. A system for according to claim 4, wherein:

said application view client passes the service request to the distributed request queue in a request document.

10. A system according to claim 9, wherein:

said application view further passes a service response listener with the request document, the service response listener adapted to listen for the service response corresponding to the service request document.

11. A system according to claim 1, wherein:

said service processor is deployed on a node in a cluster.

12. A system according to claim 11, further comprising:

additional service processors, each additional service processor deployed on different node in the cluster.

13. A system according to claim 12, wherein:

the additional service processors are adapted to listen to the distributed request queue for a service request, each of the additional service processors capable of pulling the service request from the distributed request queue and invoking the service for the service request if the service processor is unavailable.

14. A system according to claim 1, wherein:

said service processor further encapsulates the service response into a service response object that is sent to the distributed response queue.

15. A method for high-availability processing of asynchronous requests in a single transaction, comprising:

receiving a service request to a distributed request queue from a client application;

pulling the service request from the request queue to a service processor and invoking a service for the service request;

receiving the service response from the invoked service to a distributed response queue and storing the service response until retrieval from a client application.

16. A method according to claim 15, further comprising:

executing the invoked service using an enterprise information system.

17. A method according to claim 15, further comprising:

deploying an additional service processor on each node of the cluster containing the service processor.

18. A method according to claim 17, further comprising:

listening to the distributed request queue using with the service processor and any additional service processors.

19. A method according to claim 15, further comprising:

packaging the service processor as a message-driven Enterprise JavaBean.

20. A method according to claim 15, further comprising:

using an application view client to send service requests and receive service responses on behalf of the client application.

21. A method according to claim 20, further comprising:

generating failure recovery semantics using the application view client.

22. A method according to claim 15, further comprising:

assigning a client identifier to the service request to be used in processing the service request and service response.

23. A method according to claim 15, wherein:

the step of sending a service request includes passing a request document and response listener to the service processor.

24. A system for high-availability processing of asynchronous requests in a single transaction, comprising:

a distributed request queue for receiving and storing a service request;

an application view client for sending the service request to the distributed request queue on behalf of a client application;

a service processor for pulling the service request from the request queue and invoking the service for the service request, the

service processor further receiving a service response for the service request from the invoked service; and

a distributed response queue for receiving the service response from the service processor and storing the service response until the service response is retrieved for the client application by the application view client.

25. A system for high-availability processing of asynchronous requests in a single transaction, comprising:

an application view client for generating a service request on behalf of a client application, the service request comprising a request document and a service response listener;

a distributed request queue for receiving the service request from the application view client and storing the service request;

a service processor for pulling the service request from the request queue and invoking the service specified in the request document, the service processor further receiving a service response for the request document from the invoked service; and

a distributed response queue for receiving the service response from the service processor and storing the service response until the service response is retrieved for the client application by the application view client, the response listener notifying the application view client when the service response is received in the distributed response queue.

26. A computer-readable medium, comprising:

means for receiving a service request to a distributed request queue from a client application;

means for pulling the service request from the request queue to a service processor and invoking a service for the service request; and

means for receiving the service response from the invoked service to a distributed response queue and storing the service response until retrieval from a client application.

27. A computer program product for execution by a server computer for high-availability processing of asynchronous requests in a single transaction, comprising:

computer code for receiving a service request to a distributed request queue from a client application;

computer code for pulling the service request from the request queue to a service processor and invoking a service for the service request; and

computer code for receiving the service response from the invoked service to a distributed response queue and storing the service response until retrieval from a client application.;

28. A system for high-availability processing of asynchronous requests in a single transaction, comprising:

means for receiving a service request to a distributed request queue from a client application;

means for pulling the service request from the request queue to a service processor and invoking a service for the service request; and

means for receiving the service response from the invoked service to a distributed response queue and storing the service response until retrieval from a client application.

29. A computer system comprising:

a processor;

object code executed by said processor, said object code configured to:

receive a service request to a distributed request queue from a client application;

pull the service request from the request queue to a service processor and invoking a service for the service request; and

receive the service response from the invoked service to a distributed response queue and storing the service response until retrieval from a client application.

30. A computer data signal embodied in a transmission medium, comprising:

a code segment including instructions to receive a service request to a distributed request queue from a client application;

a code segment including instructions to pull the service request from the request queue to a service processor and invoking a service for the service request; and

a code segment including instructions to receive the service response from the invoked service to a distributed response queue and storing the service response until retrieval from a client application.

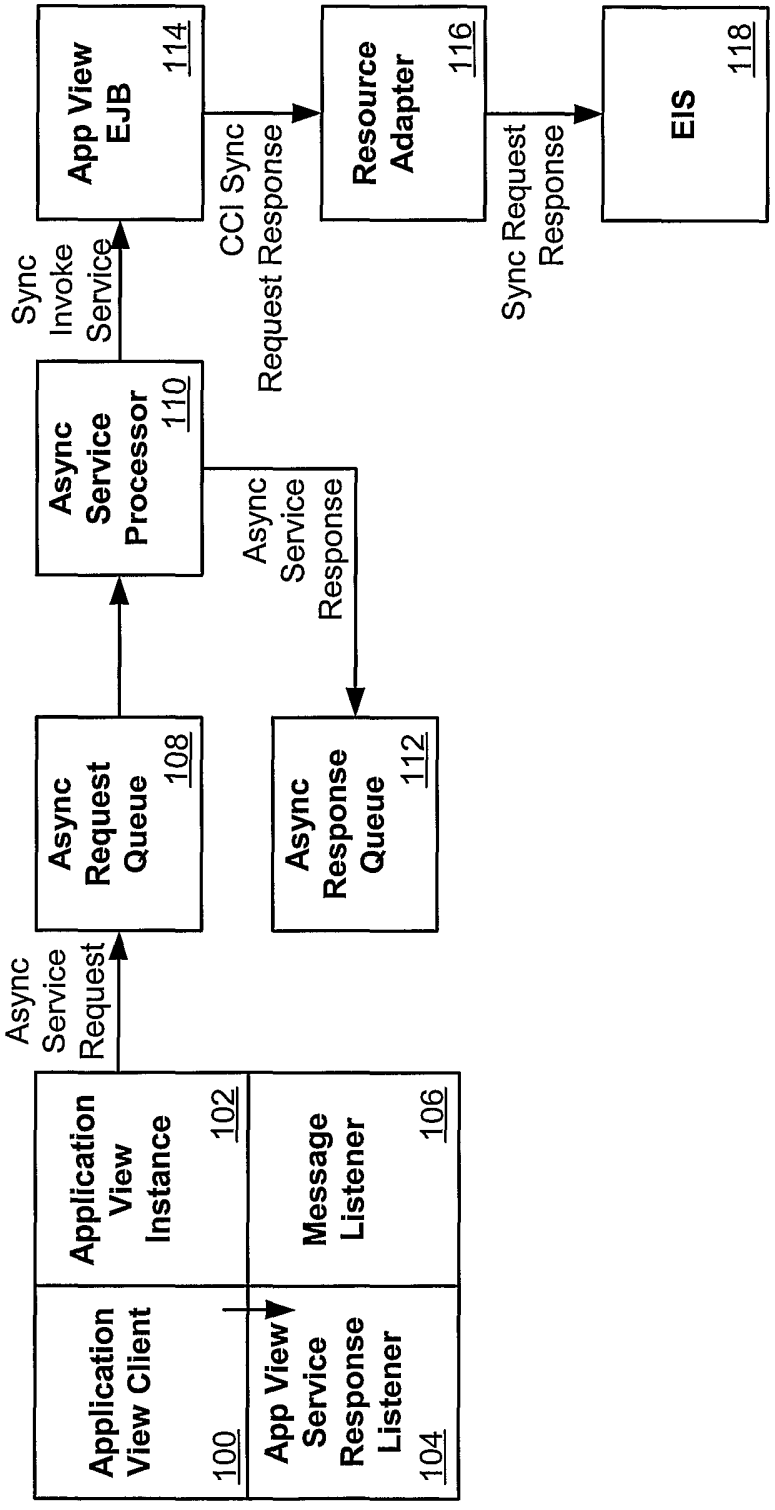

Application View Client
100
Application View Instance
102
App View Service Response Listener
104
Message Listener
106
Async Service Request
Async Request Queue
108
Async Response Queue
112
Async Service Response
Async Service Processor
110
Sync Invoke Service
App View EJB
114
CCI Sync Request Response
Resource Adapter
116
Sync Request Response
EIS
118

Figure 1

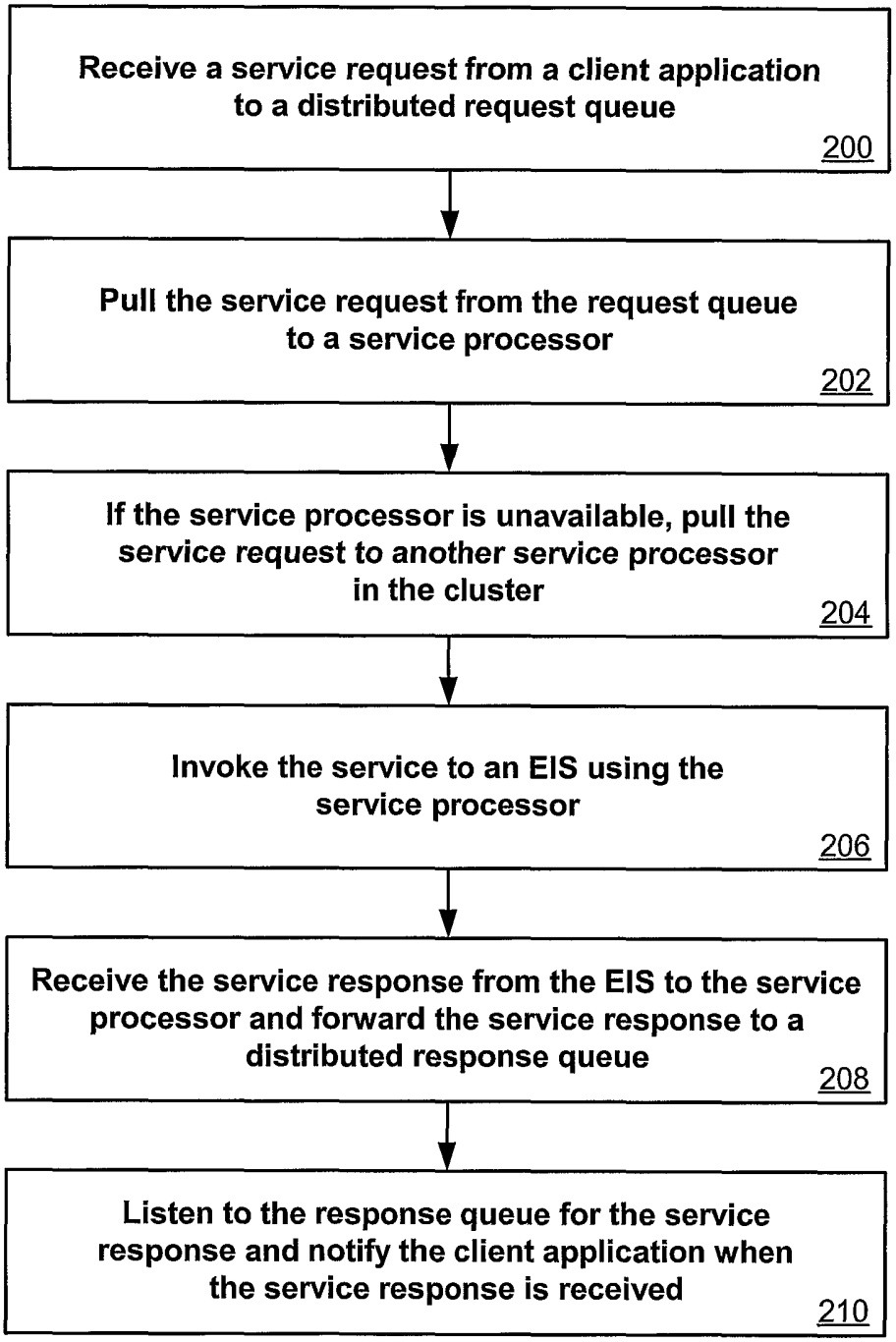

Receive a service request from a client application to a distributed request queue
200
Pull the service request from the request queue to a service processor
202
If the service processor is unavailable, pull the service request to another service processor in the cluster
204
Invoke the service to an EIS using the service processor
206
Receive the service response from the EIS to the service processor and forward the service response to a distributed response queue
208
Listen to the response queue for the service response and notify the client application when the service response is received
210

Figure 2

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US03/13621

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) :G06F 15/16, 13/14
US CL :709/ 200-203, 217-219, 227-29, 245; 707/3, 9-10
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/ 200-203, 217-219, 227-29, 245; 707/3, 9-10

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
NONE

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

WEST
Search Terms—> asynchronous request, transaction, queue, distributed, service processor, invok$, java, client application

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y, P	US 6,553,425 B1 (SHAH et al) 22 April 2003. abstract, column 2 line 42 to column 3 line 38, column 4 line 60 to column 5 line 6, column 10 line 50 to column 11 line 40.	1-30
Y, P	US 6,442,611 B1 (NAVARRE et al) 27 August 2002. abstract, figures 2-3, column 2 line 25 to column 3 line 64, column 7 line 26 to column 8 line 24.	1-30
Y	US 6,154,769 A (CHERKASOVA et al) 28 November 2000. abstract, column 2 line 55 to column 3 line 18, column 4 lines 46-63, column 6 lines 4-56.	1-30

[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 07 JULY 2003	Date of mailing of the international search report 08 AUG 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 BHARAT BAROT Telephone No. (703) 305-4092

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

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US03/13621

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A —	US 6,012,094 A (LEYMANN et al) 04 January 2000.	1-30