PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H04L 29/08, G06F 9/46

A1

(11) International Publication Number: **WO 98/34386**

(43) International Publication Date: 6 August 1998 (06.08.98)

(21) International Application Number: PCT/US98/01644

(22) International Filing Date: 29 January 1998 (29.01.98)

(30) Priority Data:
08/794,269 3 February 1997 (03.02.97) US

(71) Applicant: ORACLE CORPORATION [US/US]; 500 Oracle Parkway, Redwood Shores, CA 94065 (US).

(72) Inventors: ADUNUTHULA, Seshu; 34542 Felix Terrace, Fremont, CA 94555 (US). ANAND, Mala; 501 Forest Avenue, Palo Alto, CA 94301 (US). CHOU, Tsung-Jen; 15 La Loma Drive, Menlo Park, CA 94025 (US). NAKHODA, Shehzaad; Apartment #14, 455 Grant Avenue, Palo Alto, CA 94306 (US). NG, Raymond; Apartment #3, 1111 Foster City Boulevard, Foster City, CA 94404 (US). PANG, Robert; Apartment #19, 1240 Dale Avenue, Mountain, CA 94040 (US). SHARMA, Ankur; Apartment #365, 951-2 Old County Road, Belmont, CA 94002 (US). BOOKMAN, Matthew; 14855 La Rinconada Drive, Los Gatos, CA 95030 (US).

(74) Agent: CARLSON, Stephen, C.; McDermott, Will & Emery, Suite 300, 99 Canal Center Plaza, Alexandria, VA 22314 (US).

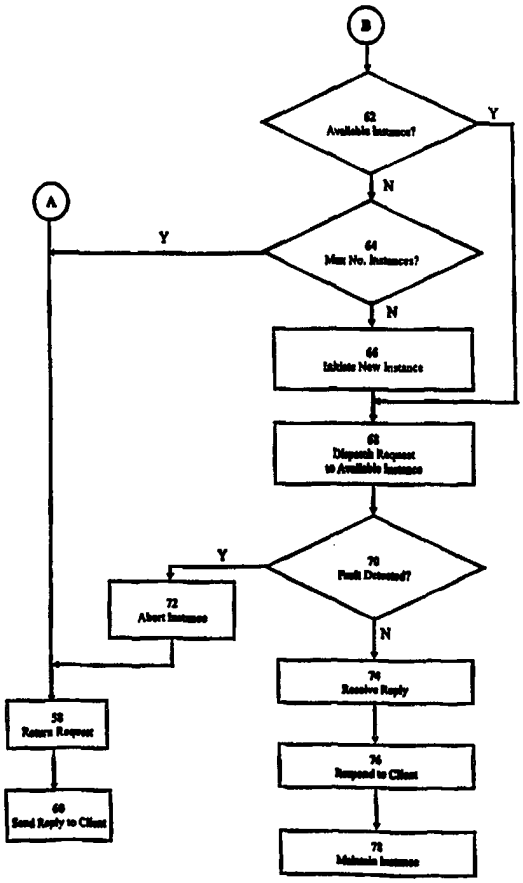
(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, GW, HU, ID, IL, IS, JP, KE, KG, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: WEB REQUEST BROKER CONTROLLING MULTIPLE PROCESSES

(57) Abstract

A web server configured to respond to client requests over a network such as the World Wide Web includes a web listener having a Hypertext Transfer Protocol (HTTP) daemon, a plurality of extension programs configured to perform respective operations, and a web request broker configured to identify one of the programs for responding to a client request, and determine the availability of an instance of the identified program. The web request broker maintains control of multiple instances of each server extension program to provide enhanced server operation without overwhelming server resources. The web request broker maintains a minimum number of instances of the identified program in memory, each executed in its own address space. The web request broker determines whether an available instance of the identified program is available from an existing number of instances, and selectively initiates a new instance of the program if no other instance is available. If no instance is available and the existing number of instances exceeds the maximum prescribed number, then the web request broker returns the reply to the web listener to send a reply over the network that the request was not processed.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

WEB REQUEST BROKER CONTROLLING MULTIPLE PROCESSES

FIELD OF THE INVENTION

This invention relates to server architectures in networked computer systems, more specifically to web servers executing server applications supporting dynamic operations for web users.

BACKGROUND OF THE INVENTION

The World Wide Web includes a network of servers on the Internet ("web servers"), each of which has one or more HTML (Hypertext Markup Language) pages. The HTML pages on a web server provide information and hypertext links to other documents on that and (usually) other web servers. Web servers communicate with clients by using the Hypertext Transfer Protocol (HTTP).

Users of the World Wide Web use a client program, referred to as a browser, to request, decode and display information from a selected web server. When the user of a browser selects a link, a request is sent over the Internet to the web server that stores information specified in the link. In response to the request, the web server transmits the specified information to the browser that issued the request. The browser receives the information, presents the received information to the user, and awaits the next user request.

Traditionally, the information stored on web servers is in the form of static HTML pages. Static HTML pages are created and stored at the web server prior to a request from a web browser. In response to a request, a static HTML page is merely read from storage and transmitted to the requesting browser. Currently, there is a trend to develop web server applications that respond to browser requests by performing dynamic operations. For example, a web server may respond to a request by issuing a query to a database, dynamically constructing a web page containing the results of the query, and transmitting the dynamically constructed HTML page to the requesting browser. To perform dynamic operations, the functionality of the web server must be enhanced or augmented. Various approaches have been developed for extending web servers to support dynamic operations.

One approach to the provide dynamic operations in response to requests from web browsers uses the common gateway interface (CGI). CGI is a specification for transferring

information between a web server and a CGI program. A CGI program is any program designed to accept and return data that conforms to the CGI specification. The program could be written in any programming language, including C, Perl, or Visual Basic.

The CGI approach suffers from the disadvantage that a separate process (a separate instance of the CGI program) is initiated each time the specified request is received by the server. Receipt of a thousand such requests from different users will thus cause a thousand processes to be initiated, exhausting available resources on the server.

An alternative approach to providing dynamic responses to requests involves using a "plug-in" extensions. A plug-in extension intercepts messages sent to the server at various stages to perform application-specific processing for a specific user request. A web server plug-in executes in the same address space as the web server and all other web server plug-ins. Hence, an application developer designing a plug-in must be familiar with the lower level operational details of the web server. Moreover, execution of the plug-ins in the same address space as the web server exposes the web server to security and stability risks, where a faulty plug-in may cause other plug-ins or the web server itself to crash, or perform in an unpredictable manner.

SUMMARY OF THE INVENTION

There is a need for an arrangement that enables web servers to support dynamic server operations, where multiple external processes may be initiated, managed, and terminated in a controllable, scalable and efficient manner.

There is also a need for an arrangement for responding to a client request issued to a web server executing multiple instances of a program configured to process the request, where the request from the client is selectively dispatched to an available instance.

There is also a need for an arrangement that responds to a client request, where an instance of a program configured to process the request is selectively initiated based on the availability of existing instances and a predetermined maximum number of instances.

These and other needs are attained by the present invention, where a web request broker controls processing of a request by identifying a program that corresponds to the request, selectively initiating an instance of the program, and dispatching the request to the instance to process the request.

According to one aspect of the present invention, a request issued to a server from a client over a network system is processed by obtaining the request over a network, and identifying a program that corresponds to the request. An instance of the program is selectively initiated based on a prescribed number of instances of the program. The request is dispatched to the initiated instance of the program, and the instance executes the corresponding program to process the request. The request is then responded to, based on the execution of the instance. Hence, a plurality of programs, for example server extensions, may be added to a server process in a controllable manner. Moreover, the selective initiation of an instance of an identified program ensures that the server maintains control of the multiple instances, preserving stability in server operations. The prescribed number of instances may also specify both a minimum and a maximum number of instances, enabling processing delays to be minimized by maintaining at least a minimum number of instances in memory for subsequent requests, while maintaining control of server resources by limiting the maximum number of instances.

According to another aspect of the present invention, a method for execution by a server is configured to respond to a request for performance of an operation. The method includes obtaining the request over a network, and forwarding the request to a dispatcher plug-in executed by the server. The request is processed by causing the dispatcher plug-in to determine whether an available instance of a program, configured to handle the request, is available from an existing number of the program instances. If an instance is available, then the request is dispatched for execution by the available instance. If no instance is available, then a new instance is initiated if the existing number of instances does not exceed a maximum prescribed number. If no instance is available and the existing number of instances exceeds the maximum prescribed number; then a reply is sent over the network indicating the request was not processed. Hence, the dispatcher plug-in manages server resources in processing the request by selectively dispatching the request for execution or denying the request based upon the availability of an instance relative to the maximum prescribed number of instances.

Hence, the present invention enables a plurality of extension programs to be added to a server process in a controllable manner. The dispatcher plug-in controls execution of different extension programs running in separate and independent instances, and selectively

routes requests to available instances, ensuring that the server process and the server extension programs are not overloaded.

Additional objects, advantages and novel features of the invention will be set forth in part in the description which follows, and in part will become apparent to those skilled in the art upon examination of the following or may be learned by practice of the invention. The objects and advantages of the invention may be realized and attained by means of the instrumentalities and combinations particularly pointed out in the appended claims.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings in which like reference numerals refer to similar elements and in which:

Figure 1 is a block diagram of a web server responding to a request received from a client over a network system, according to an embodiment of the present invention;

Figure 2 is a block diagram of the web server according to a first embodiment of the present invention;

Figures 3A and 3B are flow diagrams summarizing the method for responding to the client request according to an embodiment of the present invention;

Figure 4 is a flow diagram illustrating a method of initiating the server process according to an embodiment of the present invention; and

Figure 5 is a block diagram illustrating the web request broker according to a second embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

A method and apparatus for responding to a request issued to a server from a client over a network system is described. In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that the present invention may be practiced without the specific details. In other instances, well-known structures and devices are shown in block diagram form in order to avoid unnecessarily obscuring the present invention.

OVERVIEW OF WEB SERVER ARCHITECTURE

Figure 1 is a diagram of a web server responding to a request received from a client over a network system according to an embodiment of the present invention. The web server 10 receives the request from a client 12 over a network system 14, for example the World Wide Web using Transmission Control Protocol/Internet Protocol (TCP/IP). The web server 10 includes a web listener 16, a web request broker 18, and a plurality of server extension programs 20. A user of the network 14 uses the client program 12 to request, decode and display information from the web server 10. The client program 12 includes a web browser 22 that sends a request to the web listener 16 via the network 14. The client program also includes browser extension programs 24 (e.g., "plug-in" extensions) that provide additional processing capabilities for the web browser 22. Communication between the web browser 22 and the web listener 16 is executed using standardized protocol, for example Hypertext Transfer Protocol (HTTP), Version 1.0. The HTTP 1.0 protocol may be used with optional secure sockets layer (SSL) based data-encryption to establish a short-term connection between the web browser 22 and the web listener 16.

As described below, the web listener 16 receives the client request over the network 14, and forwards the request to the web request broker 18. The web request broker 18 selectively dispatches the request to an executable instance of one of the server extension programs 20 for processing. The web listener 16, upon receiving a reply from the web request broker 18, outputs the reply to the client request via the network 14. Upon receiving the reply, the web browser 22 determines the type of request received, and determines how to handle the response. For example, the response may either be processed natively by the web browser 22, or the web browser 22 may use one of the browser extension programs 24 for further processing. The browser extension programs 24 will typically be implemented as a client-side plug-in that performs specific processing of the reply. Upon completing the processing, the client 12 will typically display the result in the web browser's main viewing area as a hypertext mark-up language (HTML) page.

WEB REQUEST BROKER

According to the present invention, web request broker 18, is configured to manage processing of client requests by selectively routing the client request to server extensions running in separate processes.

Figure 2 is a block diagram of the server 10 according to a first embodiment of the present invention. The web listener 16 includes an HTTP daemon 16a that supports network transport according to HTTP protocol. The web listener 16 receives the client request from the network 14, typically delivered in the form of a Uniform Resource Locator (URL). The client request serves as an identifier for a web object, for example an HTML page or an operation to be performed. The web listener 16 hands off the client request to the web request broker 18 without any attempt at interpreting the client request.

The web request broker 18 includes a dispatcher plug-in 30 and a plurality of execution engines 32. The web request broker 18 controls processing of the client request by identifying an extension program 20 configured to process the client request, and dispatching the client request for execution by an available instance of the extension program. The dispatcher plug-in 30 includes a configuration library 34 that identifies the available programs for handling different requests, described in detail below. Once the dispatcher plug-in 30 identifies a program extension 20 that is configured to process the request, the dispatcher plug-in 30 determines whether an available instance of the program configured to handle the request is available, and dispatches the request for execution by the available instance, described below.

The web server 10 also includes a plurality of server extension programs 20a, 20b and 20c. Each server extension program, also referred to as a system cartridge, is configured for a different operation. Specifically, a server extension program is configured as a cartridge that performs a well-defined function, or as a programmable cartridge that acts as an interpreter or a routine environment for an application. An example of a programmable cartridge is a PL/SQL agent 20a, configured to process database queries according to the Oracle-based Programming Language using Structured Query Language (PL/SQL). The PL/SQL agent 20a executes a client request having a database query by executing an individual process 36 (i.e., a separate instance of the PL/SQL agent 20a). Execution of the instance 36a causes the instance to process the request, for example accessing a database server 40 in communication with the instance 36 via a data link 42.

Another example of a programmable cartridge-type server extension program is a JAVA interpreter 20b, which enable web application developers to write server-side JAVA applications to process client requests. Similarly, a custom server 20c may be configured as

an extension program in order to provide dynamic operations, for example accessing processes executed by a third party server 46.

The extension programs 20a, 20b, and 20c, stored as executable code, are executed by first initiating an instance 36 of the corresponding extension program 20 into server memory, and executing the instance. An instance is equivalent to a process in a UNIX environment. The web request broker 18 manages the execution of each of the extension programs 20 by initiating a predetermined minimum number of instances 36a, 36b, 36c for the extension programs 20a, 20b, 20c, respectively. If the web request broker 18 receives a client request and determines that no instance 36 of the appropriate extension program is available, the web request broker 18 will initiate a new instance of the program to execute the request if the existing number of instances does not exceed a maximum prescribed number.

For example, if a client request specifies a request for access of the database 40, the web request broker 18 will identify the PL/SQL agent 20a as the program configured to handle the request. The web request broker 18 will determine whether an existing instance 36a of the program 20a is available to handle the request. If no instance is available, e.g., all the existing instances $36a_1$-$36a_n$ are processing other client requests, the web request broker 18 will initiate a new instance $36a_{n+1}$ if the existing number of instances 36a does not exceed a maximum prescribed number.

As shown in Figure 2, the web request broker 18 includes web request broker execution engines (WRBX) 32 for each of the extension programs 20. The execution engine 32 controls execution of the instances of the corresponding program by providing an application programming interface (WRB API) that specifies predetermined operations to be performed by the instances of the corresponding program. By establishing basic callback functions between the execution engine 32 and an extension program, any extension program can be integrated into the server 10 by configuring the extension program to respond to the callback functions (for example an initialization function, a request handler, and a shutdown function), and then registering the extension program in the configuration library 34, described below.

Thus, if the dispatcher plug-in 30 determines that the PL/SQL agent 20a is the appropriate extension to process a request, the dispatcher plug-in 30 dispatches the request to the execution engine 32a. If a new instance of the program 20 needs to be initiated, the

dispatcher plug-in 30 creates a new instance of the program in a separate address space and dispatches the request to the execution engine 32a of the new instance. The address space used to execute the instance of the program may be within memory of the computer system upon which the web request broker is executing, or on another computer system. The execution engine 32a then issues a request handler callback function to the specified instance $36a_i$, causing the instance $36a_i$ to process the request, for example by accessing the database 40. The instance $36a_i$ executing the request returns the result to the execution engine 32a, which forwards the result to the dispatcher plug-in 30. In the event that the web request broker 18 detects a fault in the operation, the execution engine 32a issues a shutdown function to abort the instance from memory.

Hence, the execution engine 32a provides an application programming interface to the web request broker 18 (WRB API) that specifies predetermined operations to be performed. Use of the WRB API enables programmers of the extension programs 20 to configure each extension program for high-level integration into the server 10 independent of the protocols used by the particular web listener with which the extension program will be used.

Figures 3A and 3B summarize a flow diagram illustrating a method of responding to the client request according to an embodiment of the present invention. The client request is received in step 50 by the web listener 16. Upon receiving the client request, the web listener 16 forwards the request to the web request broker 18 in step 52. The dispatcher plug-in 30 identifies the program that corresponds to the client request by accessing in step 54 the configuration library 34. The configuration library 34 includes for each program an object type corresponding to the request processed by the corresponding program. For example, if the client request is a URL request beginning with the virtual path "/java", the configuration library 34 will store a corresponding object specifying that the JAVA interpreter 36b is configured to handle requests having the virtual path "/java". The configuration library 34 will also include a virtual path specifying an address location for the stored program used to initiate instances of the program 20.

The dispatcher plug-in 30 determines in step 56 if the request object type (e.g., the virtual path specified in the client request) corresponds to an identifiable program, where the request object type corresponds to an object type stored in the configuration library 34. If the request object type does not correspond to an identifiable program, the request is returned to

the web listener 16 in step 58 (see Figure 3B). If in step 58 the HTTP daemon 16a recognizes the request as a request for a static HTML page, the HTTP daemon accesses the static HTML page from the page memory 16b, and sends the reply to the client in step 60. If the client request is not recognized by the HTTP daemon, the reply is sent to the client in step 60 indicating that the request was unrecognizable.

If in step 56 the dispatcher plug-in 30 identifies from the configuration library 34 an extension program configured to handle the request, the dispatcher plug-in 30 determines in step 62, shown in Figure 3B, whether an available instance of the identified program is available among the existing number of instances 36. If in step 62 the dispatcher plug-in 30 identifies an available instance, for example instance $36a_2$ of the PL/SQL agent 20a, the corresponding execution engine 32 is called in step 68 to execute the available instance to process the request, described below. However, if in step 62 no instance of the identified program 20 is available, the dispatcher plug-in 30 determines in step 64 if the existing number of instances exceeds a maximum prescribed number, stored in the configuration library 34. If the existing number of instances exceeds the maximum prescribed number in step 64, the dispatcher plug-in 30 returns the request to the web listener 16 in step 58, after which the web listener sends a reply to the client over the network in step 60 indicating the request was not processed.

If in step 64 the existing number of instances does not exceed the maximum prescribed number, the dispatcher plug-in 30 initiates a new instance of the identified program and dispatches the request to the execution engine 32a of the new instance. For example, the dispatcher plug-in 30 initiates a new instance of the PL/SQL agent 20a. During this step, the stored sequences of instructions for the PL/SQL agent 20a are accessed to create a new instance $36a_i$ of the program 20a in an address space that is separate from the address space in which dispatcher plug-in 30 is executing.

Once the new instance $36a_i$ is running, the dispatcher plug-in 30 dispatches the request to the execution engine 32a associated with the new instance $36a_i$ in step 68. The execution engine 32a sends a callback message to the new instance $36a_i$ requesting execution of the request. The execution engine 20 passes in the callback message any parameters necessary for the instance $36a_i$ to process the request, for example passwords, database search keys, or any other argument for a dynamic operation executed by the instance $36a_i$. The instance $36a_i$ then executes the request. During the execution of the request by the

instance in step 68, the dispatcher plug-in 30 monitors the instance to determine the occurrence of a fault in step 70. If in step 70 the dispatcher plug-in 30 detects a fault, the dispatcher plug-in 30 calls the corresponding execution engine 32 in step 72 to abort the instance 36 having the fault. The corresponding execution engine 32 in turn issues a shut down command across the API to the faulty instance. The instance, responding to the shut down command by the execution engine 32, will shut down without affecting any other process in any other address space.

If in step 70 no fault is detected, the dispatcher plug-in 30 receives a reply from the instance 36 upon completion of execution in step 74. The dispatcher plug-in 30 in step 76 forwards the reply to the web listener 16, which responds to the client with the reply from the executed instance 36. After executing the instance, the dispatcher plug-in 30 in step 78 maintains the instance in the memory, as shown in step 78 to enable execution of a subsequent request.

Hence, the disclosed arrangement manages multiple instances of different extension programs to process a variety of user requests. Each instance 36 for any program 20 is executed in a separate memory space, enabling a faulty instance 36 of a program 20 to be aborted without affecting any other instances of the programs. The web request broker 18 also controls the number of instances for each given extension program 20. Hence, server resources are controlled to ensure that a large number of requests do not overwhelm the server 10 by an uncontrollable generation of instances. Execution throughput also is improved by maintaining a minimum number of instances ready for execution. Moreover, additional instances may be initiated and maintained in memory for executing subsequent requests, as opposed to terminating an instance after a single execution and then reloading the extension program into memory in order to recreate an instance for execution of a subsequent request.

Figure 4 is a diagram illustrating the initialization of the server 10 according to an embodiment of the present invention. The server 10 is initialized by starting the server process in step 90, where the web listener and supporting processes are loaded into memory space. The server 10 then starts the dispatcher plug-in 30 in step 92, causing the sequences of instruction for executing the web request broker 18 to be stored in memory. The extension programs 20 are then registered with the dispatcher plug-in 30 in step 94 by storing in the configuration library 34, for each extension program 20: (1) the cartridge name; (2) the

minimum number of required instances 36; (3) the maximum number of instances; (4) the virtual path for accessing the extension program, i.e., the address space to be accessed to initiate a new instance of the program; (5) the program-dependent function names used by the execution engine to execute the callback functions (initialization, request handler, shutdown); and (6) an object identifier, for example an object type to be supplied by a client request for requesting performance of an operation by the corresponding extension program 20. The object type may be a specific word, or may include a virtual path, for example "/java". The extension programs 20 may be registered in step 94 by a server manager, i.e., a web master having access to the configuration library in a real-time user-interactive environment. Once the server manager has established the configuration library, the extension programs may be registered in step 94 automatically by accessing a non-volatile memory, for example a disk.

After registering the extension programs with the dispatcher plug-in 30, the dispatcher plug-in 30 initiates the minimum instances for each program in a separate address space in step 96. Once the minimum number of instances has been initiated, the server 10 is prepared to process client requests. Each execution engine 32 tracks the location in memory and status of each instance 36 of the corresponding program 20.

Figure 5 is a block diagram of the server 10, according to a second embodiment of the present invention. The first embodiment of Figure 2 assumes that the dispatcher plug-in 30 is compatible with the lower level processes of the web listener 16 and the HTTP daemon 16a. The embodiment of Figure 5 is a modification of the first embodiment of Figure 2, in that the server includes a transport adapter 17 that receives a client request from the HTTP daemon 16' operating according to a protocol different from the web request broker 18.

The transport adapter 17 is configured to recognize the protocols of different HTTP daemons, and can convert the client requests received from the HTTP daemon 16' into a converted client request having a second protocol independent from the protocol of the HTTP daemon 16' and matching the protocol of the web request broker 18. Hence, the transport adapter 17 enables the web request broker 18 to be used with HTTP daemons from different vendors. Moreover, transport adapter 17 may be configured to accommodate different server architectures and operating systems. Hence, the transport adapter 17 converts a client request from the HTTP daemon 16' from a first protocol to a second protocol compatible with the web request broker 18. Similarly, replies from the web request

broker are converted to the transport protocol of the HTTP daemon 16' to enable the HTTP daemon 16' to send the reply to the user via the network.

In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than restrictive sense.

What is claimed is:

1. A method for responding to a request issued to a server from a client over a network system, the method comprising the steps of:

obtaining the request over a network;

identifying a program that corresponds to the request;

selectively initiating an instance of the program based on a prescribed number of instances of said program;

dispatching the request to said instance of said program;

executing said instance to cause said instance to process said request; and

responding to the request based on execution of said instance.

2. The method of Claim 1, wherein:

the step of obtaining the request over a network is performed by a server process executing in a first address space;

the step of selectively initiating an instance of the program includes initiating said instance in a second address space that is separate from the first address space.

3. The method of Claim 1, further comprising the step of, after executing said instance, maintaining said instance in memory for executing a subsequent request.

4. The method of Claim 1, further comprising the steps of:

detecting a fault in the instance during execution thereof; and

aborting the instance in response to detecting said fault.

5. The method of Claim 1, wherein the prescribed number of said instances specifies at least one of a maximum and a minimum number of instances, wherein each instance executes within an address space that is separate from the address spaces used by other instances of said program.

6. The method of Claim 5, wherein:

the step of selectively initiating an instance of the program includes the step of initiating a specified minimum number of instances of said program prior to obtaining the request over the network; and

the method further includes the step of, after obtaining the request over the network, determining if one of the minimum number of instances is available for processing said request.

7. The method of Claim 6 further including the step of, after obtaining the request over the network, initiating a new instance of the program if none of the minimum number of instances is available for processing said request and the number of instances is less than the maximum number of instances.

8. The method of Claim 6, wherein the step of executing said instance includes executing an available instance from the minimum number of instances.

9. The method of Claim 1, wherein the step of selectively initiating an instance of the program includes the step of returning the request to the network without processing said request based on the prescribed number of said instances exceeding a maximum value.

10. The method of Claim 1 further comprising the step of registering a plurality of programs with the server, wherein registration of each of said programs includes specifying a maximum number of instances for said program, and a virtual path for said program.

11. The method of claim 10, wherein:

the request identifies a virtual path; and

the step of identifying a program that corresponds to the request includes the step of identifying the program based on the virtual path identified in the request.

12. The method of Claim 1, wherein:

the step of obtaining the request over a network comprises receiving the request from a transport protocol process operating according to a first protocol; and

the method includes the step of converting the request to a second protocol independent from the first protocol.

13. A method, for execution by a server, for responding to a request for performance of an operation, the method comprising:

obtaining the request over a network;

forwarding the request to a dispatcher executed by the server; and

processing the request by causing the dispatcher to perform the steps of:

A) determining whether an available instance of a program configured to handle the request is available among an existing number of instances of the program,

B) if an available instance is available, then dispatching the request for execution by the available instance,

C) if no instance is available, then initiating a new instance of the program for execution of the request if the existing number of instances does not exceed a maximum prescribed number, and

D) if no instance is available and the existing number of instances exceeds the maximum prescribed number, then sending said reply over the network indicating the request was not processed.

14. The method of Claim 13, further comprising the step of:

receiving, by the dispatcher, a reply from the instance executing the request; and

sending information contained in the reply over the network from the dispatcher to a client that issued the request.

15. The method of Claim 13, further comprising the step of:

detecting by the dispatcher a fault in the instance executing the request; and

terminating by the dispatcher the instance executing the request in response to the detected fault.

16. The method of Claim 15, further comprising the step of sending a reply over the network indicating the request was not processed.

17. The method of Claim 13, further comprising the step of registering with the dispatcher a plurality of programs configured to handle respective types of requests, the step of registering with the dispatcher a plurality of said programs including the step of storing in the dispatcher for each of said programs a maximum number of instances and a virtual path specifying an address location associated with the corresponding program.

18. The method of Claim 13, wherein each instance is executed within an address space that is separate from the address spaces used by other instances of said program.

19. The method of Claim 18, wherein the step of initiating a new instance of the program comprises the step of initiating the new instance within an address space that is separate from the address spaces used by the other instances of said program.

20. The method of Claim 19, further comprising the step of delaying deallocation of said new instance at least a predetermined time interval after processing the request for processing a subsequent request.

21. The method of Claim 13, wherein:

the step of obtaining the request over the network comprises receiving the request from a transport protocol process operating according to a first protocol; and

the method further comprises the step of converting the request to a second protocol independent from the first protocol.

22. The method of Claim 13, wherein the existing number of said instances is at least a prescribed minimum number of said instances.

23. The method of Claim 13, further comprising the steps of:

causing the dispatcher to determine if a program is configured to handle the request based on the operation specified in the request; and

if no program is configured to handle the request, then sending a reply over the network indicating the request was not processed.

24. The method of Claim 13, wherein:

the step of obtaining the request over the network comprises the step of executing by the server a server process; and

the step of processing the request by causing the dispatcher to perform said steps comprises the step of executing by the server a plug-in routine, added to the server process, to cause the dispatcher to perform said steps.

25. A computer readable medium having stored thereon sequences of instructions for responding to a request for performance of an operation received by a server, the sequences of instructions including instructions for performing the steps of:

obtaining the request over a network;

identifying a program that corresponds to the request;

selectively initiating an instance of the program based on a prescribed number of instances of said program;

dispatching the request to said instance of said program;

executing said instance to cause said instance to process said request; and

responding to the request based on execution of said instance

26. The computer readable medium of Claim 25, further comprising sequences of instructions for performing the steps of:

detecting a fault in the instance during execution thereof; and

aborting the instance in response to detecting said fault.

27. The computer readable medium of Claim 25, wherein the prescribed number of said instances specifies at least one of a maximum and a minimum number of instances, wherein each instance executes within an address space that is separate from the address spaces used by other instances of said program.

28. A computer server configured to respond to a request for performance of an operation, comprising:

a network listener configured to receive the request over a network and send a response to the request over the network, the request having a prescribed object type specifying an operation to be performed;

a plurality of programs, each program configured to perform an operation that generates an output in response to receiving a request having a corresponding object type specifying the operation performed by the program, each program having a prescribed number of instances executing at respective address spaces; and

a dispatcher plug-in configured to identify one of the programs for responding to the request based on the prescribed object type, the dispatcher plug-in selectively dispatching the request to an available instance of the identified one program based upon the corresponding prescribed number of instances, the dispatcher plug-in sending said response to the network listener based on execution of the request by the available instance.

29. The server of Claim 28, wherein the network listener receives and sends the request and the response based on Hypertext Transfer Protocol (HTTP).

30. The server of Claim 29, wherein the network listener includes an HTTP daemon configured to output a static Hypertext Markup Language (HTML) page in response to the prescribed object type specifying sending said static HTML page.

31. The server of Claim 28, further comprising a transport adapter configured to convert the request received by the network listener from a first protocol to a second protocol independent from the first protocol, the transport adapter converting the response output by the dispatcher plug-in from said second protocol to the first protocol before sending by the network listener.

32. The server of Claim 28, further comprising a plurality of execution engines, each execution engine configured to control execution of the instances of the corresponding programs and provide an Application Programming Interface (API) specifying predetermined operations to be performed by the instances of the corresponding program, the execution engine corresponding to the identified one program receiving the request from the dispatcher plug-in and controlling execution of the available instance processing the request.

33. The server of Claim 32, wherein the predetermined operations specified by the API includes at least one of initialization, execution of the request, and shutdown by at least one of the instances of the corresponding program.

34. The server of Claim 28, wherein each execution engine initiates a prescribed minimum number of said instances of the corresponding program.

35. The server of Claim 28, wherein the dispatcher plug-in includes a configuration library identifying for each of the programs the corresponding object type and the prescribed number of instances.

36. The server of Claim 28, wherein the object type of the request includes a virtual path specifying the identified one program.

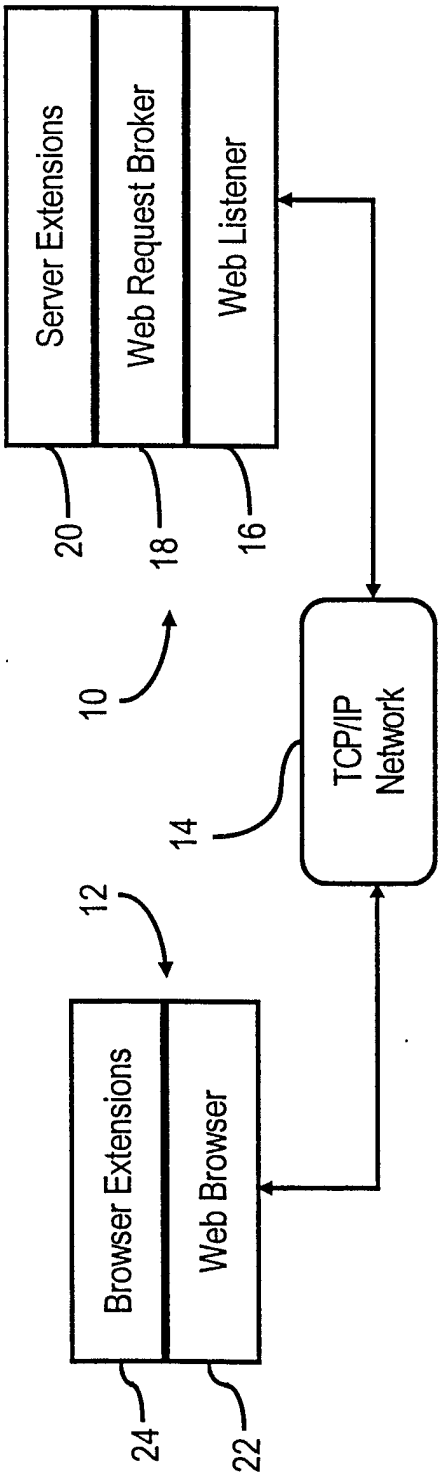

Server Extensions
Web Request Broker
Web Listener
20
18
16
10
14
TCP/IP Network
12
Browser Extensions
Web Browser
24
22

Figure 1

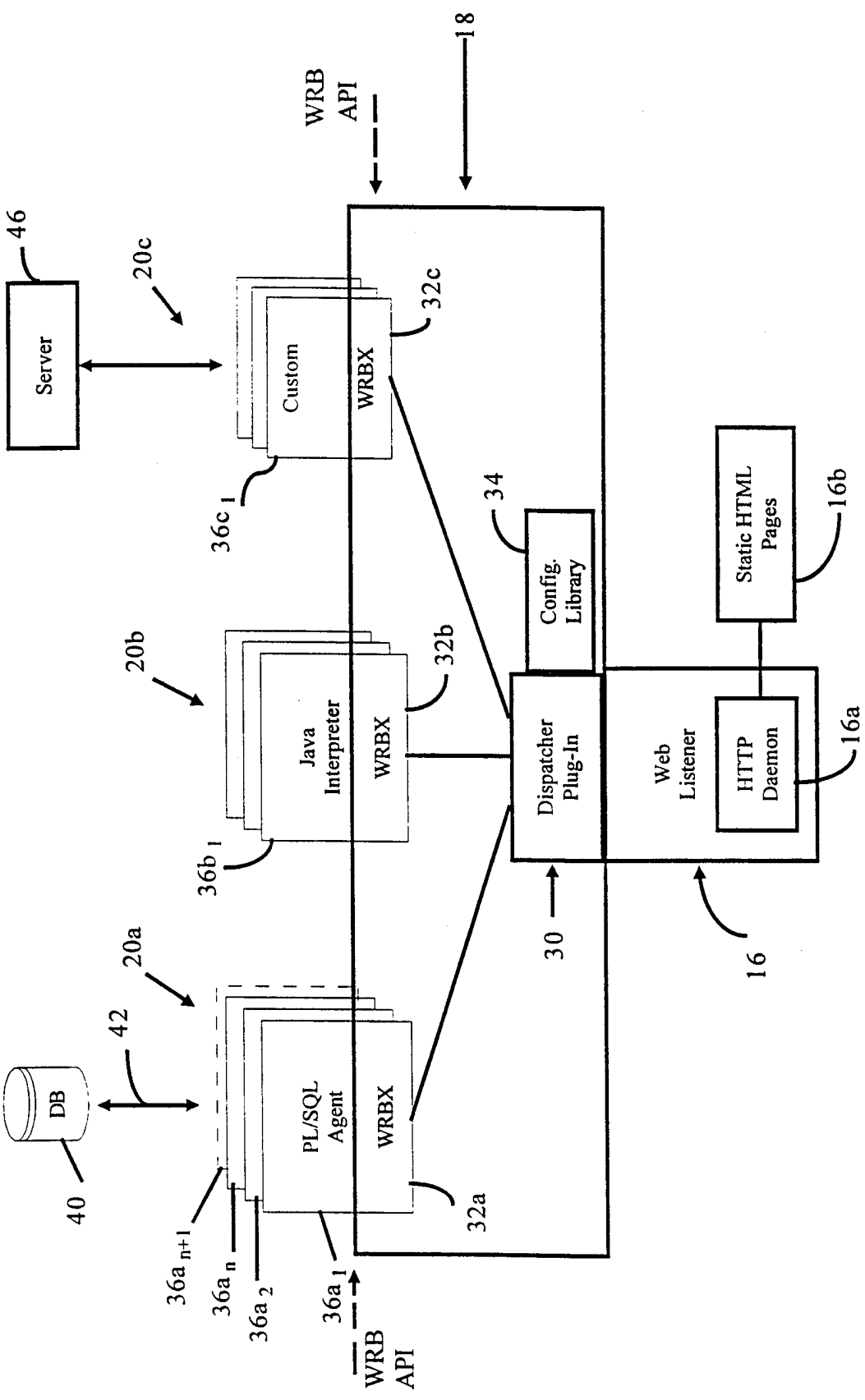

WRB API
18
46
Server
20c
Custom
WRBX
32c
36c 1
34
Config. Library
Static HTML Pages
16b
20b
Java Interpreter
WRBX
32b
36b 1
Dispatcher Plug-In
Web Listener
HTTP Daemon
16a
30
16
20a
42
DB
40
PL/SQL Agent
WRBX
32a
36a n+1
36a n
36a 2
36a 1
WRB API

Figure 2

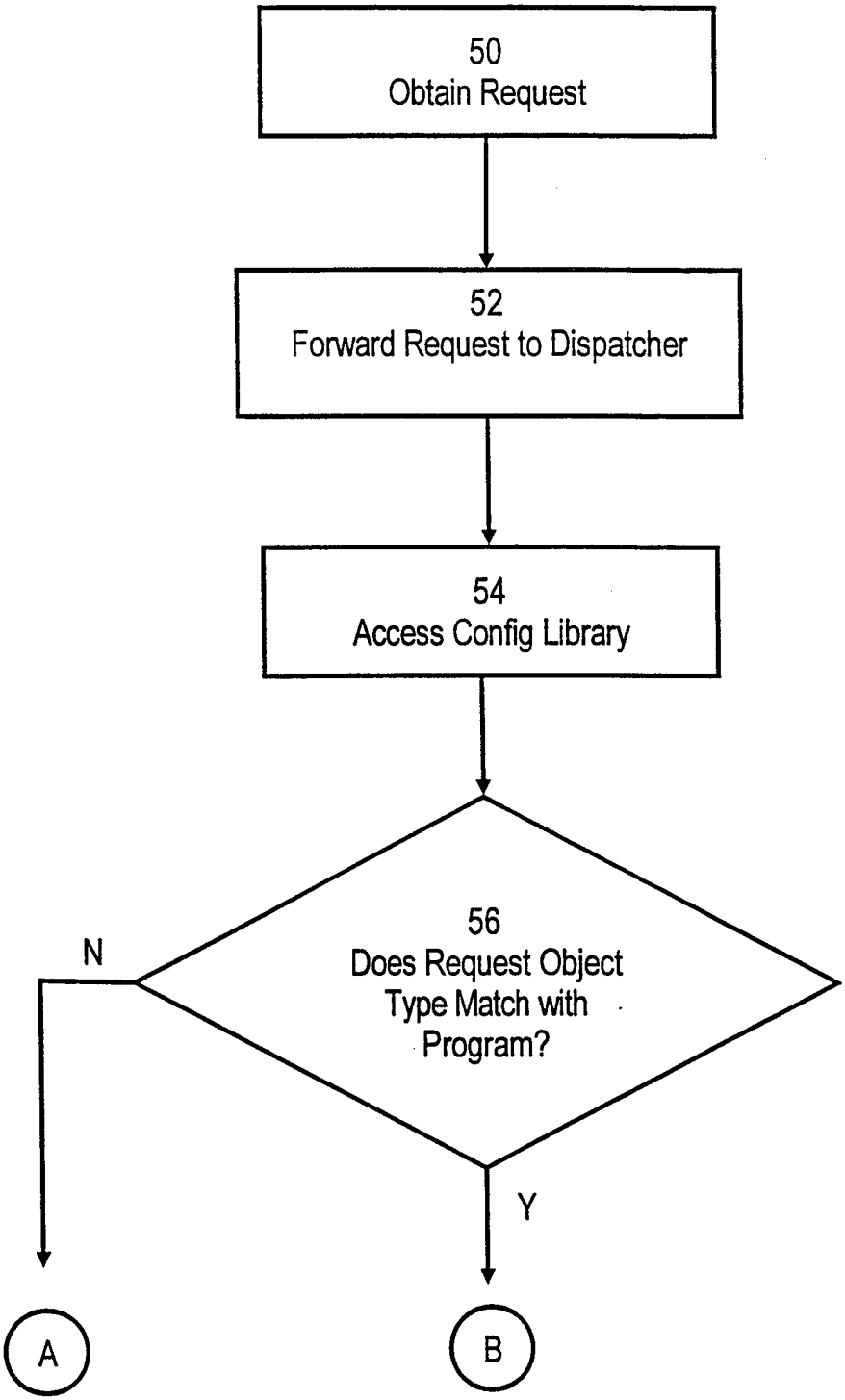

50
Obtain Request
52
Forward Request to Dispatcher
54
Access Config Library
56
Does Request Object Type Match with Program?
N
Y
A
B

Figure 3A

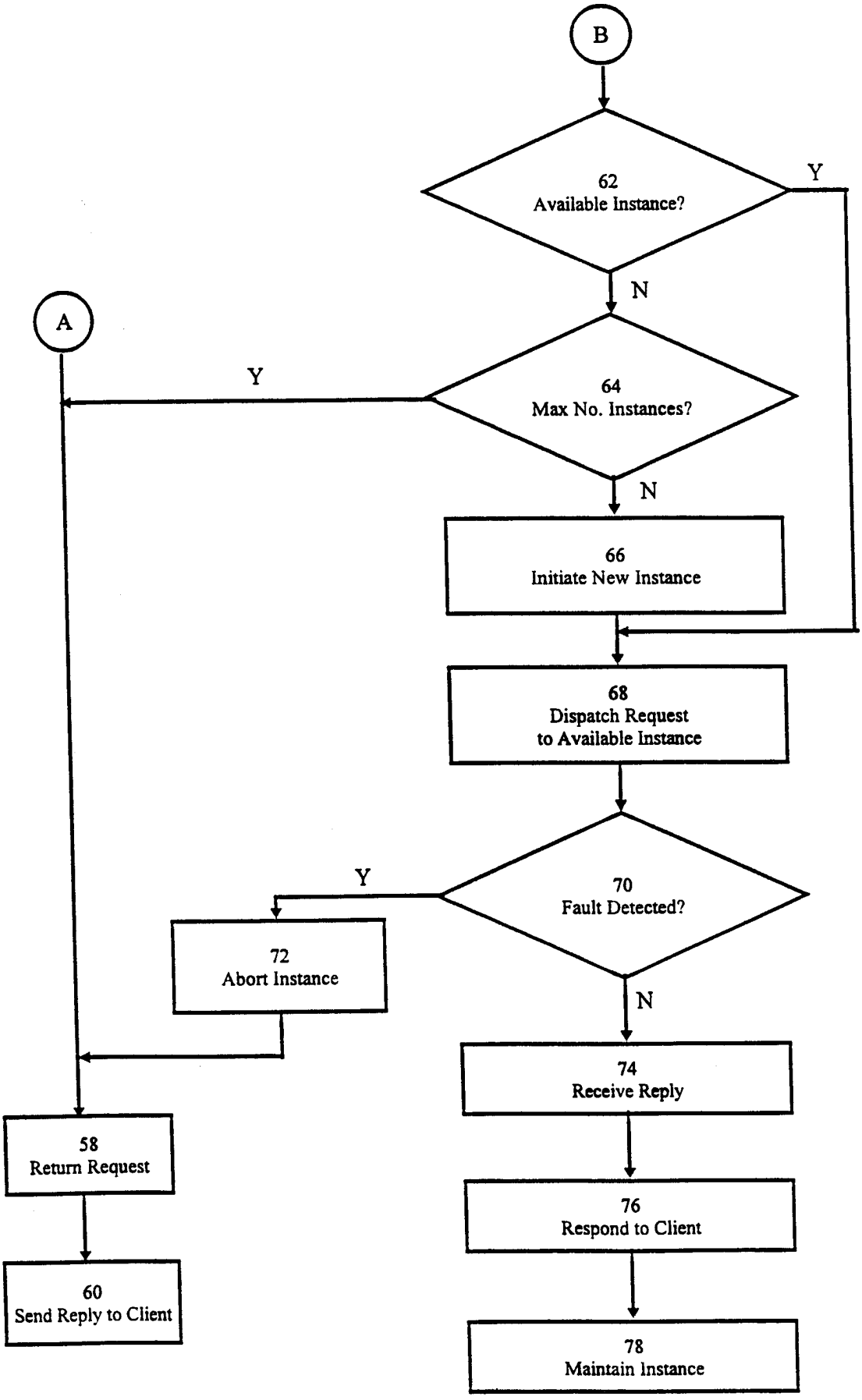

B
62
Available Instance?
Y
N
A
Y
64
Max No. Instances?
N
66
Initiate New Instance
68
Dispatch Request
to Available Instance
Y
70
Fault Detected?
72
Abort Instance
N
74
Receive Reply
58
Return Request
76
Respond to Client
60
Send Reply to Client
78
Maintain Instance

Figure 3B

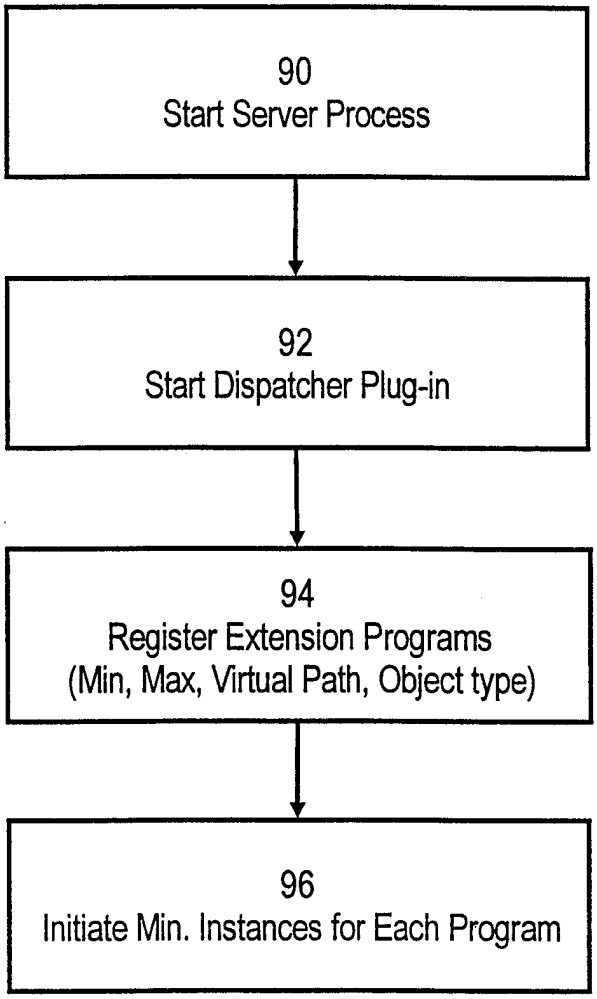

90
Start Server Process
92
Start Dispatcher Plug-in
94
Register Extension Programs
(Min, Max, Virtual Path, Object type)
96
Initiate Min. Instances for Each Program

Figure 4

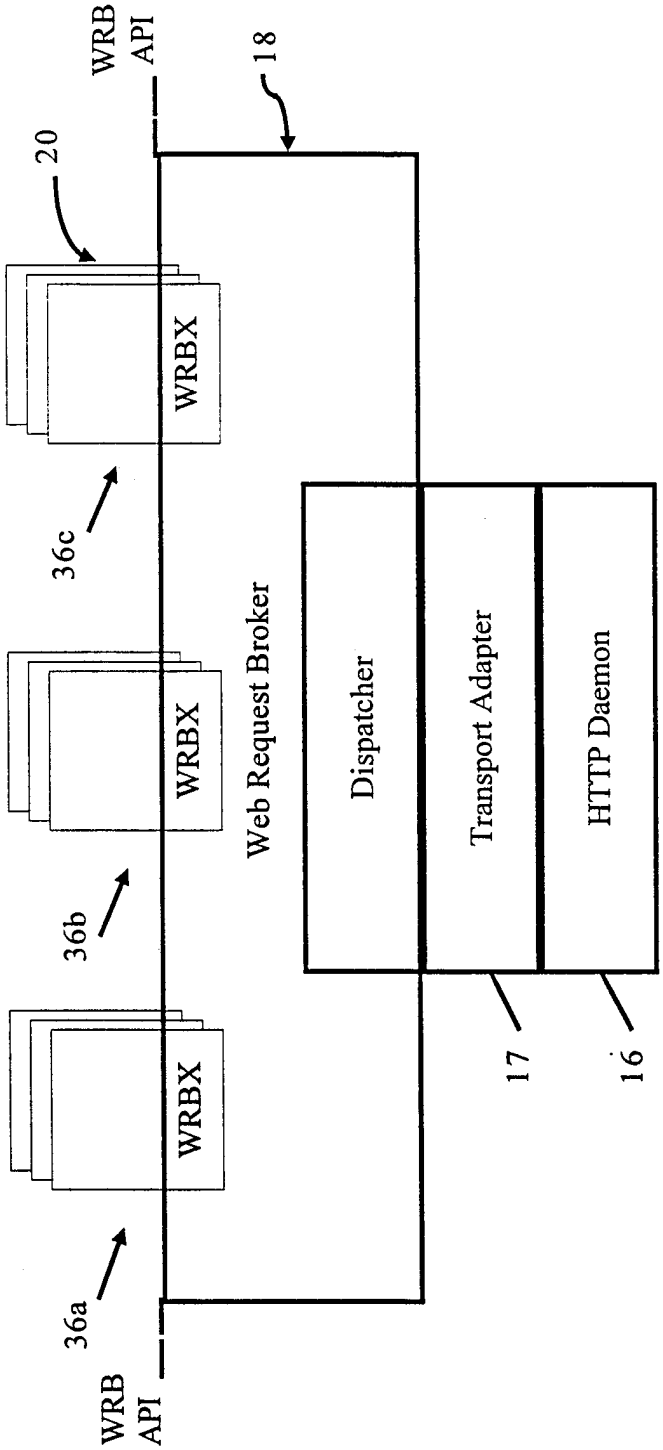

WRB API
18
20
WRBX
36c
WRBX
36b
WRBX
36a
WRB API
Web Request Broker
Dispatcher
Transport Adapter
HTTP Daemon
17
16

Figure 5

INTERNATIONAL SEARCH REPORT

Inter onal Application No
PCT/US 98/01644

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 H04L29/08 G06F9/46

According to International Patent Classification(IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 H04L G06F

Documentation searched other than minimumdocumentation to the extent that such documents are included in the fields searched

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

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	MERLE P ET AL: "CorbaWeb: A generic object navigator" COMPUTER NETWORKS AND ISDN SYSTEMS, vol. 28, no. 11, May 1996, page 1269-1281 XP004018226	1,3,4,9, 12-16, 21-26, 28-34
A	see the whole document	2,5-8, 10,11, 17-20, 27,34,36

-/--

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

[X] Patent family members are listed in annex.

° Special categories of cited documents :

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

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

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

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

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

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

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

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

"&" document member of the same patent family

Date of the actual completion of theinternational search	Date of mailing of the international search report
3 June 1998	10/06/1998

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

1

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

INTERNATIONAL SEARCH REPORT

Inter ›nal Application No
PCT/US 98/01644

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
Y	EP 0 733 969 A (SUN MICROSYSTEMS INC) 25 September 1996 see abstract see page 2, column 1, line 5 - line 14 see page 2, column 2, line 32 - line 46 see page 4, column 5, line 10 - line 13 see page 3, column 4, line 44 - line 53	1,3,4,9, 12-16, 21-26, 28-34

1

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

INTERNATIONAL SEARCH REPORT

.ormation on patent family members

Inter ›nal Application No
PCT/US 98/01644

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0733969 A	25-09-1996	CA 2171683 A	23-09-1996
		JP 8339305 A	24-12-1996