IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

ORACLE CORPORATION and ORACLE U.S.A. INC. Plaintiffs/Counterdefendants, v. EPICREALM LICENSING, LP, Defendant/Counterclaimant. AND RELATED COUNTERCLAIMS	C.A. No. 06-414 (SLR) REDACTED PUBLIC VERSION

APPENDIX OF EXHIBITS IN SUPPORT OF ORACLE'S CLAIM CONSTRUCTION BRIEF AND MOTIONS FILED ON JULY 31, 2008
VOLUME I
(A1-A35)

MORRIS, NICHOLS, ARSHT & TUNNELL LLP
Mary B. Graham (#2256)
James W. Parrett, Jr. (#4292)
1201 North Market Street
P.O. Box 1347
Wilmington, DE 19899
mgraham@mnat.com
jparrett@mnat.com
(302) 658-9200

Attorneys for Oracle Corporation and Oracle U.S.A. Inc.

OF COUNSEL:

James G. Gilliland
Theodore T. Herhold
Joseph A. Greco
Robert J. Artuz
Eric M. Hutchins
TOWNSEND AND TOWNSEND AND CREW LLP
379 Lytton Avenue
Palo Alto, CA 94301
(650) 326-2400

Dorian Daley
Peggy E. Bruggman
Matthew M. Sarboraria
ORACLE CORPORATION
ORACLE U.S.A., INC.
500 Oracle Parkway
Redwood Shores, CA 94065

DATED: July 31, 2008
Redacted Filing Date: August 7, 2008

TABLE OF CONTENTS

EXHIBIT	DOCUMENT
VOLUME I	
A1	U.S. Patent No. 5,894,554
A2	U.S. Patent No. 6,415,335
A3	Excerpts from the U.S. Patent No. 5,894,554 File History
A4	Excerpts from the U.S. Patent No. 6,415,335 File History
A5	U.S. Patent No. 6,845,505
A6	Excerpts from the U.S. Patent 6,845,505 File History
A7	U.S. Patent No. 6,334,114
A8	USPTO Order granting '554 reexam request dated May 4,2007
A9	USPTO Order granting '335 reexam request dated May 4, 2007
A10	USPTO Office Action in Ex Parte Reexamination re '554 Patent dated June 30, 2008
A11	Excerpts from W. Eckerson, "Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client/Server Applications" Patricia Seybold Group (January, 1995)
A12	Clausnitzer, et al., "A WWW Interface to the OMNIS/Myriad Literature Retrieval Engine" (April, 1995)
A13	Excerpts from the American Heritage Dictionary of the English Language, Third Edition
A14	Excerpt from the SMH Collins Dictionary of Personal Computing and the Internet
A15	General Web Terms - Oracle's Proposed Constructions Chart
A16 **REDACTED IN ITS ENTIRETY**	Excerpts from the deposition transcript of Brad Carl dated February 15, 2008

EXHIBIT	DOCUMENT
A17 **REDACTED IN ITS ENTIRETY**	Excerpts from the deposition transcript of Terry Fokas dated February 28-29, 2008
A18	Exhibit withdrawn
A19 **REDACTED IN ITS ENTIRETY**	Excerpts from the deposition transcript of Shuang Chen dated April 7, 2008
A20 **REDACTED IN ITS ENTIRETY**	Excerpts from the deposition transcript of Mark Clark dated April 17, 2008
A21 **REDACTED IN ITS ENTIRETY**	Excerpts from the deposition transcript of Steve Harris dated April 18, 2008
A22 **REDACTED IN ITS ENTIRETY**	Excerpts from the deposition transcript of Oliver Timothy Willis dated May 21, 2008
A23 **REDACTED IN ITS ENTIRETY**	Excerpts from the deposition transcript of David Finkel (DE) dated June 19-20, 2008
A24 **REDACTED IN ITS ENTIRETY**	Excerpts from the deposition transcript of Michael J. Wagner dated June 25, 2008
A25 **REDACTED IN ITS ENTIRETY**	Excerpts from the deposition transcript of David Finkel (TX) dated July 1, 2008
A26 **REDACTED IN ITS ENTIRETY**	Excerpts from the deposition transcript of Paul Clark dated July 11, 2008
A27 **REDACTED IN ITS ENTIRETY**	Excerpts from the Expert Report of Dr. David Finkel dated May 2, 2008
A28 **REDACTED IN ITS ENTIRETY**	Excerpts from the Rebuttal Report of Dr. David Finkel dated June 6, 2008
A29	Exhibit Withdrawn
A30 **REDACTED IN ITS ENTIRETY**	Expert Report of Michael J. Wagner dated May 2, 2008
A31	Curriculum Vitae of Dr. Michael Ian Shamos

EXHIBIT	DOCUMENT
A32	Joint Claim Construction and Pre-Hearing Statement (TX) dated April 21, 2006
A33	Order adopting Report and Recommendation Regarding Claim Construction (TX) dated October 30, 2006
A34	Order Denying Motion to Substitute Parties (TX) dated June 26, 2008
A35	Defendant's Response to Plaintiffs' Fourth Set of Interrogatories (Nos. 22-25) dated March 10, 2008
VOLUME II	
A36 **REDACTED IN ITS ENTIRETY**	Defendant's Supplemental Response to Plaintiffs' Interrogatory Nos. 3-5, 9-15, 17-20 and 22-25 dated May 2, 2008
A37	Parties' Proposed Element-by-Element Claim Construction Chart dated June 30, 2008
A38	Exhibit Withdrawn
A39	Excerpts from Oracle HTTP Server Administrator's Guide 10g (9.0.4)
A40	Excerpts from Oracle WebServer User's Guide (Release 1.0), dated 1995
A41	K. Montinola. "The Oracle WebServer: A Technical Discussion"
A42	Excerpts from Oracle WebServer User's Guide (Release 2.0), dated 1996
A43	Oracle WebServer Release 2.0 - The Universal Information Server for Business Applications, dated 1996
A44	Oracle WebServer 2.0 Technical Note, dated March 1996
A45	U.S. Patent No. 6,249,291 ("Popp")
A46	Excerpts from Lagoze et al., "Dienst: Implementation Reference Manual," (May 5, 1995)
A47	Declaration of Carl Lagoze dated March 29, 2007

EXHIBIT	DOCUMENT
A48	Garland et al., "Implementing Distributed Server Groups for the World Wide Web," Carnegie Mellon University Technical Report CMUCS-95-114 (Jan. 1995)
A49	Excerpts from Understanding SQL*Net, Release 2.2, Part No. A32090-1 - Oracle Corp., 1995
A50	Excerpts from Understanding SQL*Net, Release 2.3, Part No. A42484-1 - Oracle Corp., 1996
A51 **REDACTED IN ITS ENTIRETY**	Web Request Broker API dated February 12, 1996
A52	Oracle White Paper titled "Oracle and the Internet," dated July 1995
A53	Article titled, "Oracle expands World-Wide Web access; server, browser, other apps leverage database expertise; International Oracle User Group conference," PC Week (Sept. 25, 1995)
A54 **REDACTED IN ITS ENTIRETY**	Oracle Sales Records for Oracle7 Server V7.2 and SQL*Net V2.2
A55	Article titled, "Oracle Websystem Introduces Database-Enabled Webserver and Intelligent, Embeddable Web Browser," PR Newswire (Oct. 30, 1995)
A56	Oracle WebServer (Release 2.0) Feature List, dated Oct. 30, 1995
A57 **REDACTED IN ITS ENTIRETY**	Design Specification for Oracle Web Agent and OWS Developer's Toolkit, Oracle Web Server, V2.0, dated Dec. 15, 1995
A58	Press Release titled, "Oracle Announces Second Generation Web Server with Database Integration, Enhanced Security, and Server-Side Java Support" dated Feb. 21, 1996
A59	epicRealm's Supplemental Response to Oracle's Interrogatory No. 3 dated November 30, 2007
A60	Oracle Corporation Form 10-K for the fiscal year ended May 31, 2008
A61 **REDACTED IN ITS ENTIRETY**	IBM/InfoSpinner, Inc. License Agreement (Agreement Number: 4997FL3661) dated January 30, 1998

EXHIBIT	DOCUMENT
A62 **REDACTED IN ITS ENTIRETY**	Statement of Work II Services Statement of Work to License Agreement Number 4997FL3661 dated April 29, 1998
A63 **REDACTED IN ITS ENTIRETY**	Statement of Work III Services Statement of Work to License Agreement Number 4997FL3661 dated May 15, 1998
A64 **REDACTED IN ITS ENTIRETY**	Statement of Work IV Services Statement of Work to License Agreement Number 4997FL3661 dated September 29, 1998
A65 **REDACTED IN ITS ENTIRETY**	IBM Termination Letter to Bradley Carl dated October 16, 2000
A66	Excerpts from the U.S. Patent 6,334,114 File History
A67	Excerpts from the deposition transcript of B. Kent Hill dated February 28, 2008
A68	Excerpt from the Microsoft Press Computer Dictionary, Third Edition
A69	Excerpt from the IEEE Standards Dictionary of Electrical and Electronics Terms (6th ed. 1996)
A70	Excerpt from IBM Dictionary of Computing (1993)
A71	epicRealm's Preliminary Claim Constructions and Extrinsic Evidence (TX) dated March 31, 2006

2433207

A1

US005894554A

United States Patent [19]

Lowery et al.

[11] Patent Number: 5,894,554

[45] Date of Patent: Apr. 13, 1999

[54] **SYSTEM FOR MANAGING DYNAMIC WEB PAGE GENERATION REQUESTS BY INTERCEPTING REQUEST AT WEB SERVER AND ROUTING TO PAGE SERVER THEREBY RELEASING WEB SERVER TO PROCESS OTHER REQUESTS**

[75] Inventors: **Keith Lowery**, Richardson; **Andrew B. Levine**, Plano; **Ronald L. Howell**, Rowlett, all of Tex.

[73] Assignee: **InfoSpinner, Inc.**, Richardson, Tex.

[21] Appl. No.: **08/636,477**

[22] Filed: **Apr. 23, 1996**

[51] **Int. Cl.**[6] **G06F 13/14**; G06F 13/20

[52] **U.S. Cl.** **395/200.33**; 395/200.68; 395/200.75; 395/680; 707/10; 707/104

[58] **Field of Search** 358/400; 395/800, 395/700, 200.68, 200.75, 200.53, 680, 200.33; 707/104, 10

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,866,706	9/1989	Christophersen et al.	370/85.7
5,341,499	8/1994	Doragh	395/700
5,392,400	2/1995	Berkowitz et al.	395/200
5,404,522	4/1995	Carmon et al.	395/650
5,404,523	4/1995	DellaFera et al.	395/650
5,404,527	4/1995	Irwin et al.	395/700
5,452,460	9/1995	Distelberg et al.	395/700
5,532,838	7/1996	Barbari	358/400
5,751,956	5/1998	Kirsch	395/200.33
5,761,673	6/1998	Bookman et al.	707/104

OTHER PUBLICATIONS

"Beyond the Web: Excavating the Real World Via Mosaic", Goldberg et al, Second International WWW, Oct. 17, 1994.

PCT International Search Report, Aug. 21, 1997.

Primary Examiner—Thomas C. Lee

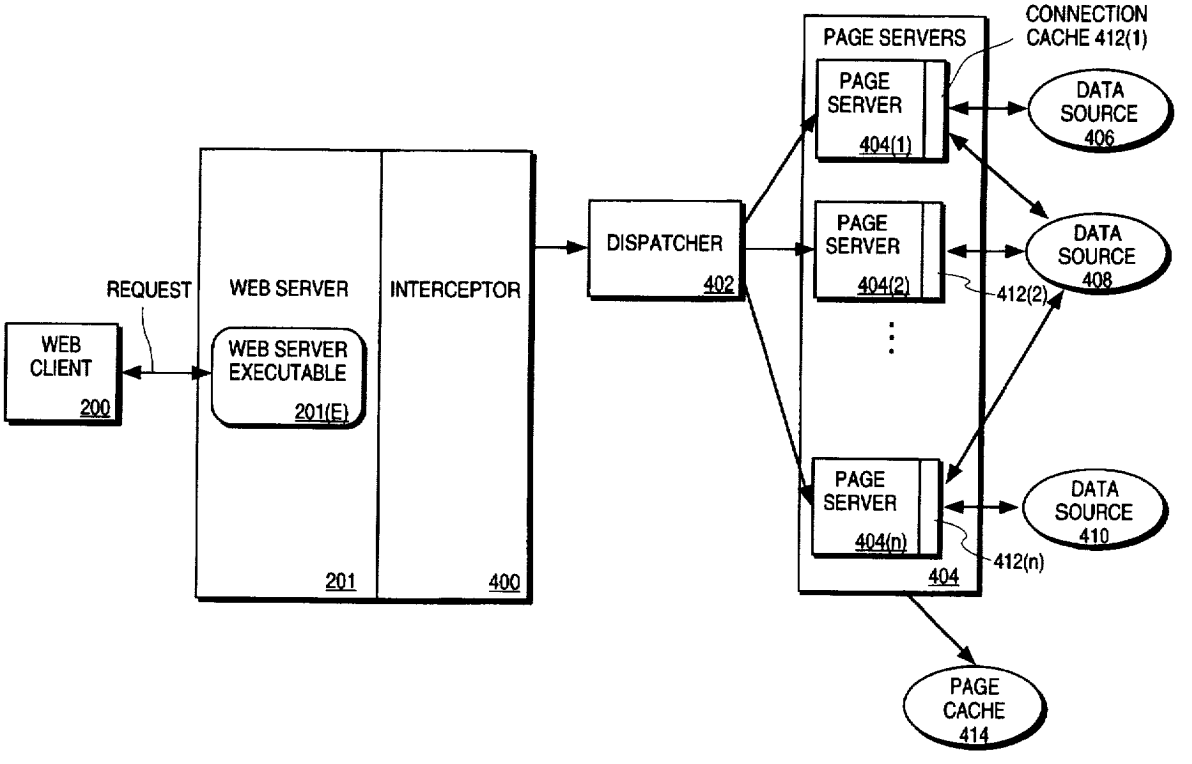
Assistant Examiner—Rehana Perveen

Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman LLP

[57] **ABSTRACT**

The present invention teaches a method and apparatus for creating and managing custom Web sites. Specifically, one embodiment of the present invention claims a computer-implemented method for managing a dynamic Web page generation request to a Web server, the computer-implemented method comprising the steps of routing the request from the Web server to a page server, the page server receiving the request and releasing the Web server to process other requests, processing the request, the processing being performed by the page server concurrently with the Web server, as the Web server processes the other requests, and dynamically generating a Web page in response to the request, the Web page including data dynamically retrieved from one or more data sources.

11 Claims, 5 Drawing Sheets



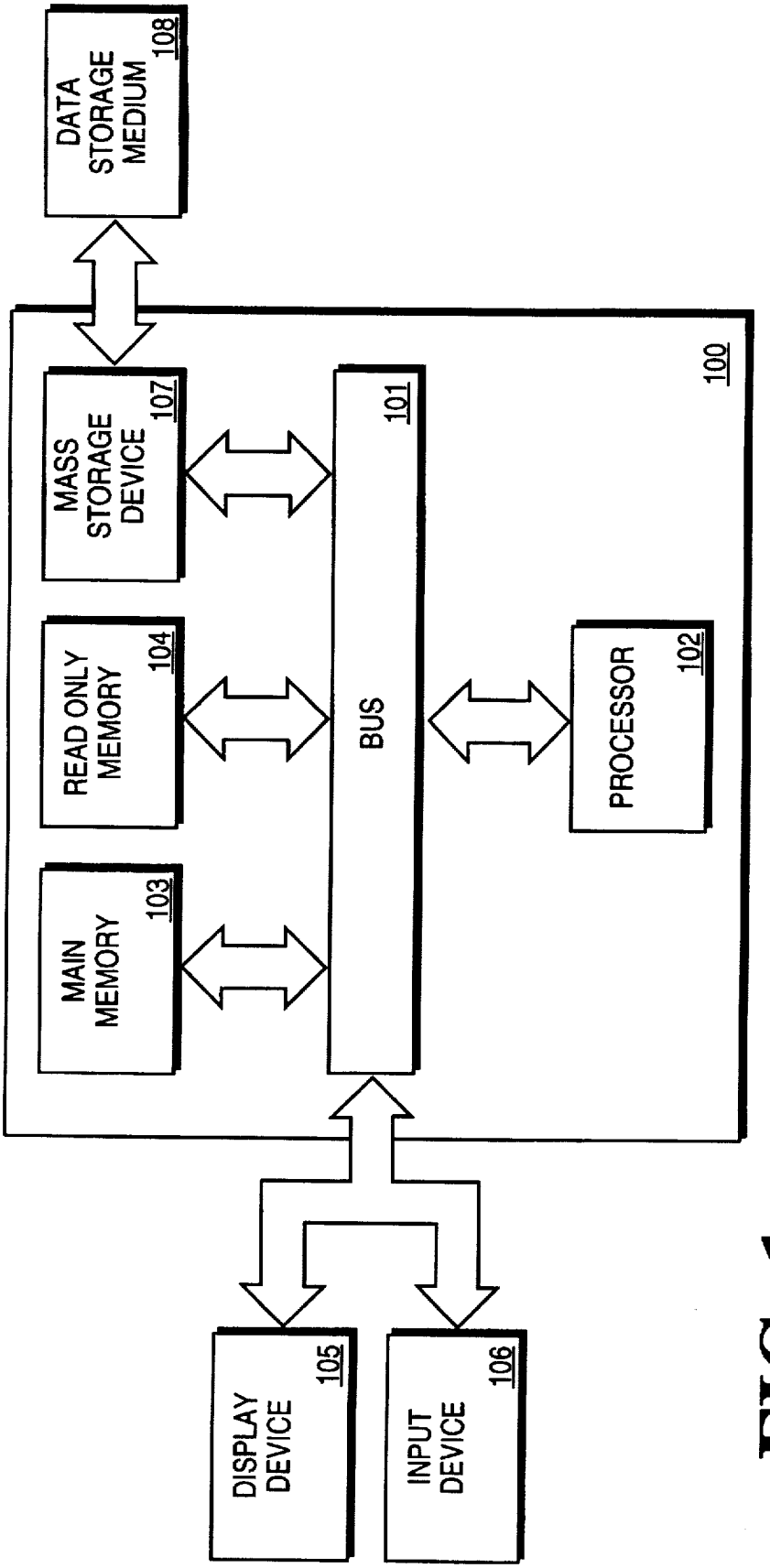

DATA STORAGE MEDIUM
108
100
MASS STORAGE DEVICE
107
101
READ ONLY MEMORY
104
BUS
PROCESSOR
102
MAIN MEMORY
103
DISPLAY DEVICE
105
INPUT DEVICE
106

FIG. 1

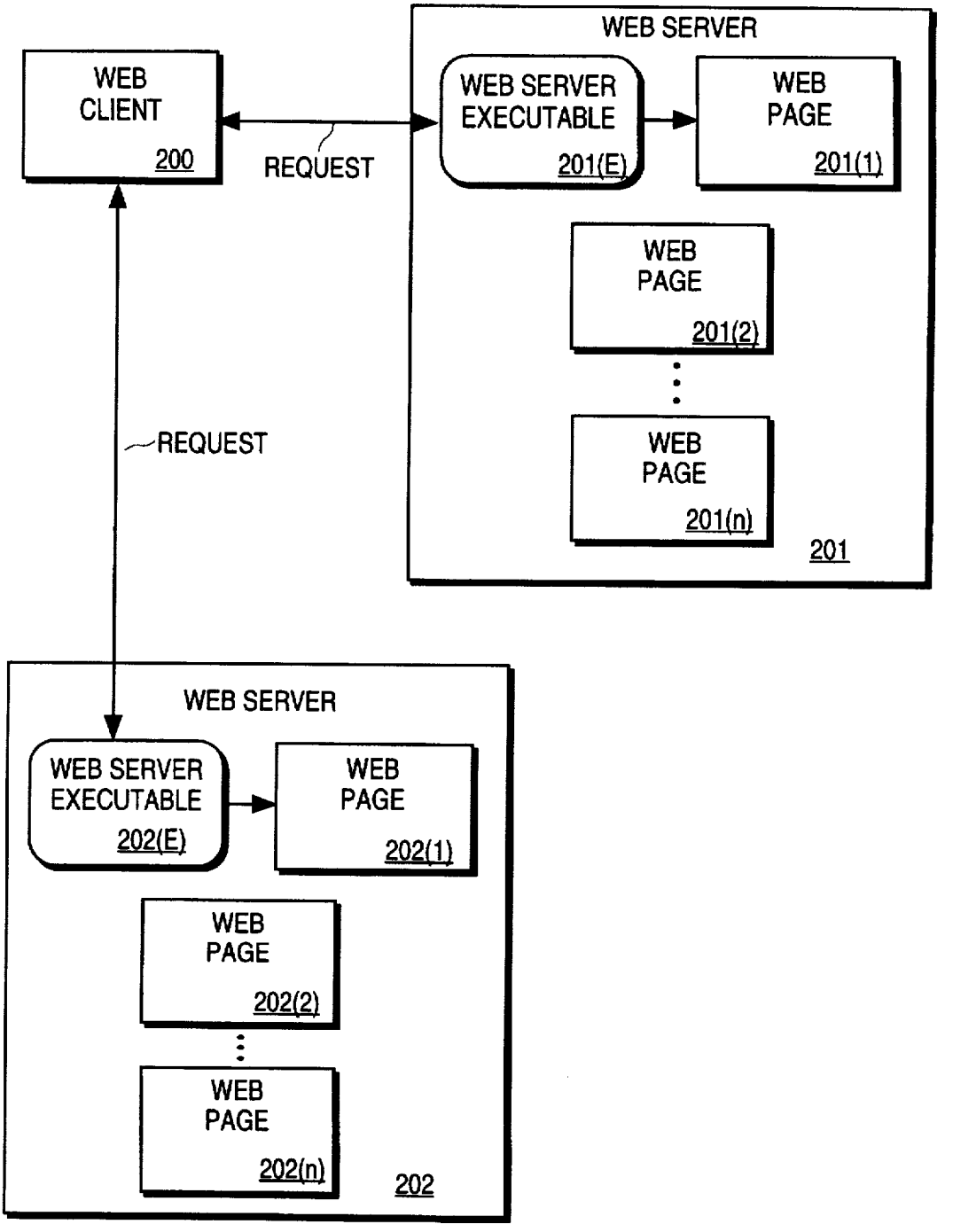


FIG. 2 (PRIOR ART)

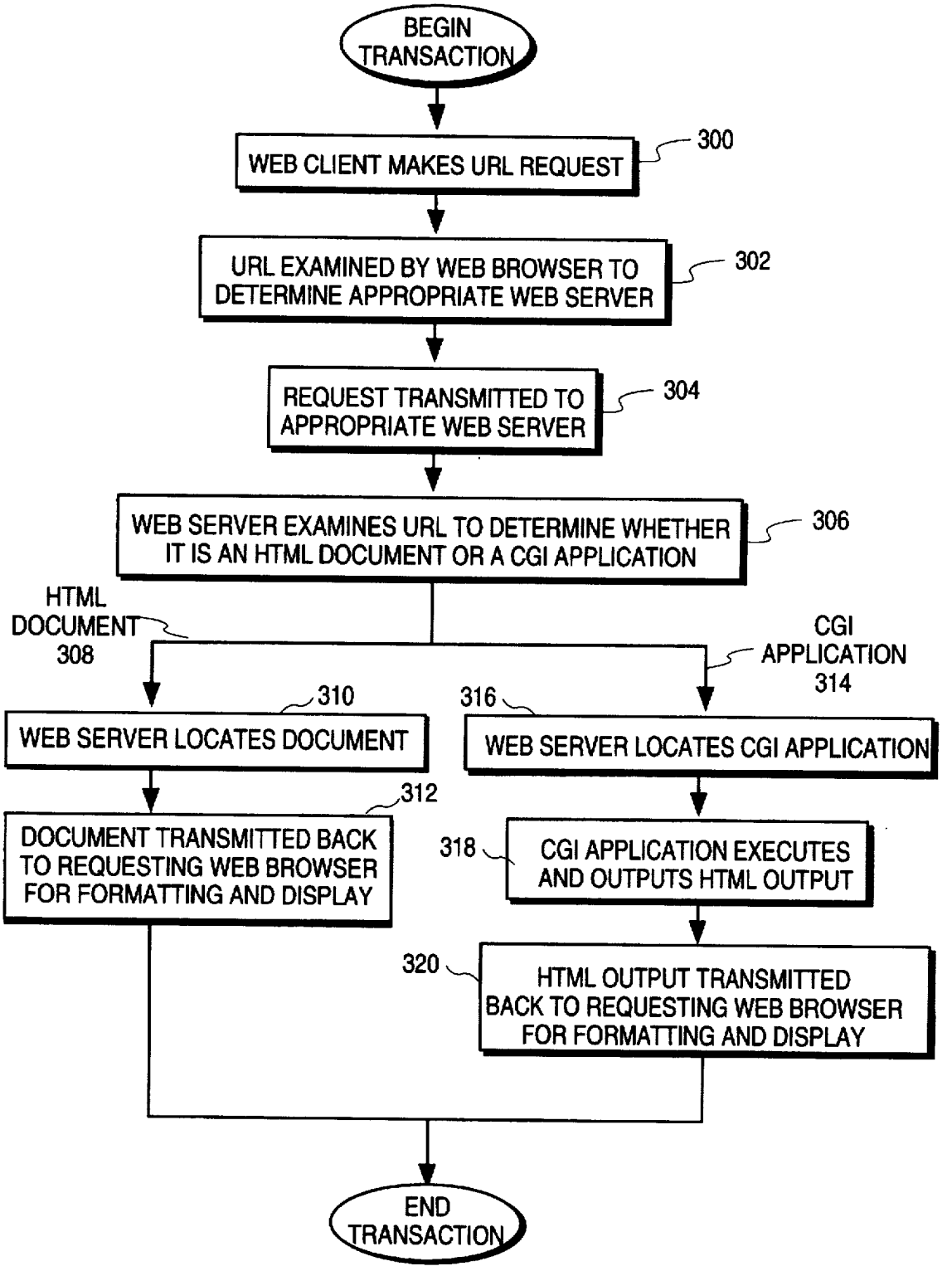


FIG. 3 (PRIOR ART)

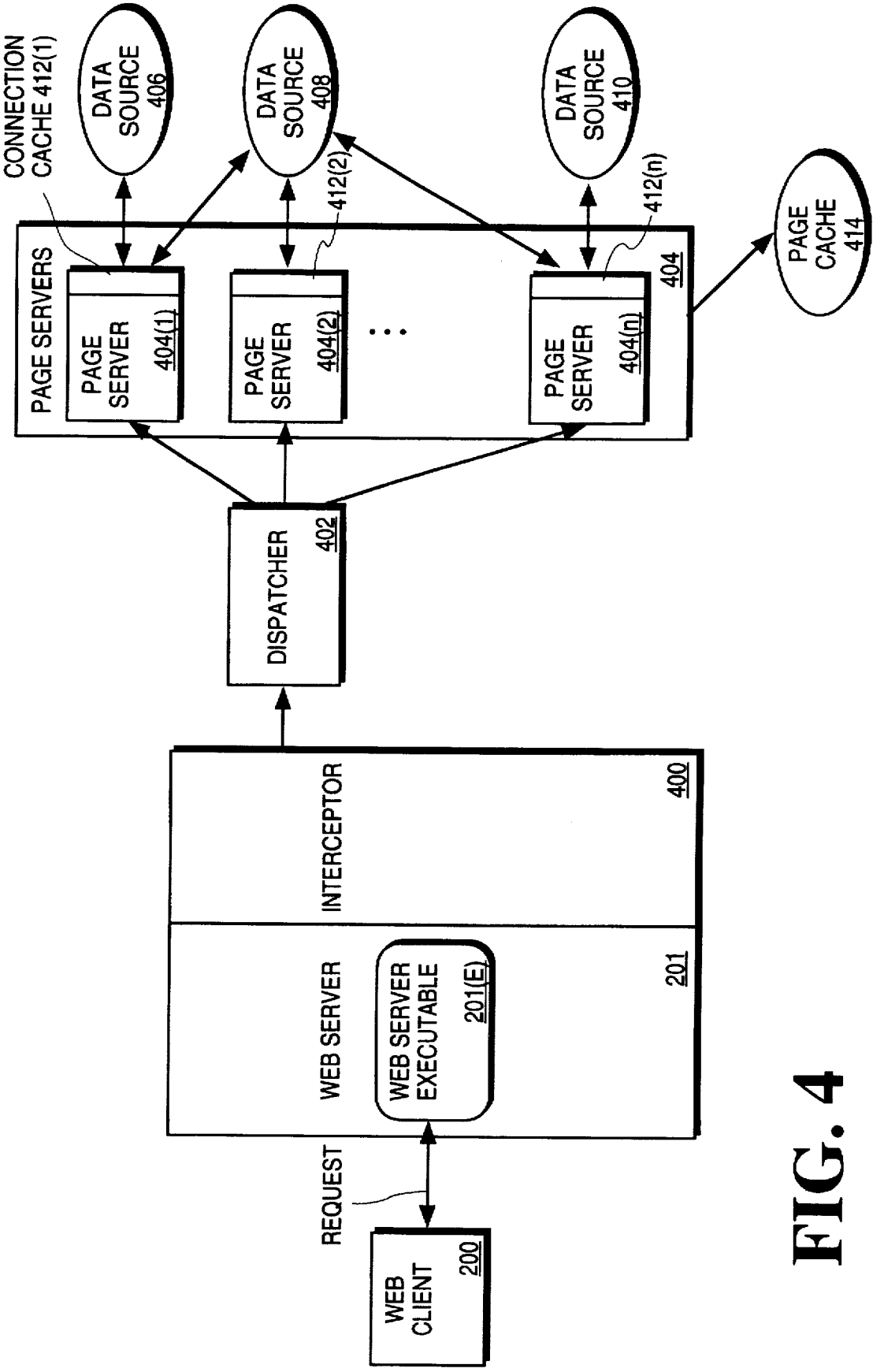

CONNECTION CACHE 412(1)
DATA SOURCE 406
DATA SOURCE 408
DATA SOURCE 410
412(2)
412(n)
PAGE CACHE 414
PAGE SERVERS
PAGE SERVER 404(1)
PAGE SERVER 404(2)
PAGE SERVER 404(n)
404
DISPATCHER 402
INTERCEPTOR 400
WEB SERVER
WEB SERVER EXECUTABLE 201(E)
201
REQUEST
WEB CLIENT 200

FIG. 4

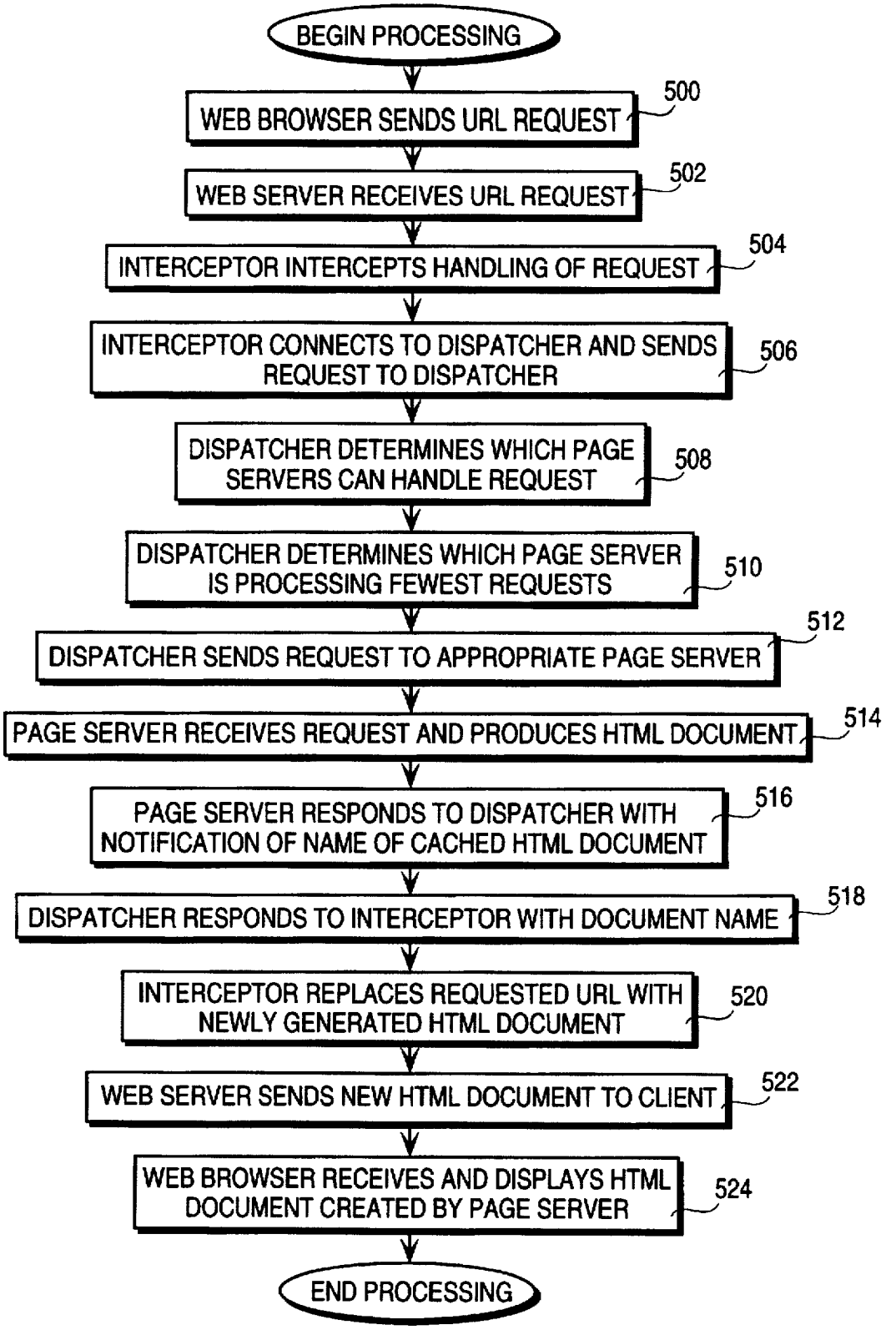

BEGIN PROCESSING
WEB BROWSER SENDS URL REQUEST
500
WEB SERVER RECEIVES URL REQUEST
502
INTERCEPTOR INTERCEPTS HANDLING OF REQUEST
504
INTERCEPTOR CONNECTS TO DISPATCHER AND SENDS REQUEST TO DISPATCHER
506
DISPATCHER DETERMINES WHICH PAGE SERVERS CAN HANDLE REQUEST
508
DISPATCHER DETERMINES WHICH PAGE SERVER IS PROCESSING FEWEST REQUESTS
510
DISPATCHER SENDS REQUEST TO APPROPRIATE PAGE SERVER
512
PAGE SERVER RECEIVES REQUEST AND PRODUCES HTML DOCUMENT
514
PAGE SERVER RESPONDS TO DISPATCHER WITH NOTIFICATION OF NAME OF CACHED HTML DOCUMENT
516
DISPATCHER RESPONDS TO INTERCEPTOR WITH DOCUMENT NAME
518
INTERCEPTOR REPLACES REQUESTED URL WITH NEWLY GENERATED HTML DOCUMENT
520
WEB SERVER SENDS NEW HTML DOCUMENT TO CLIENT
522
WEB BROWSER RECEIVES AND DISPLAYS HTML DOCUMENT CREATED BY PAGE SERVER
524
END PROCESSING

FIG. 5

SYSTEM FOR MANAGING DYNAMIC WEB PAGE GENERATION REQUESTS BY INTERCEPTING REQUEST AT WEB SERVER AND ROUTING TO PAGE SERVER THEREBY RELEASING WEB SERVER TO PROCESS OTHER REQUESTS

FIELD OF THE INVENTION

The present invention relates to the field of Internet technology. Specifically, the present invention relates to the creation and management of custom World Wide Web sites.

DESCRIPTION OF RELATED ART

The World Wide Web (the Web) represents all of the computers on the Internet that offer users access to information on the Internet via interactive documents or Web pages. These Web pages contain hypertext links that are used to connect any combination of graphics, audio, video and text, in a non-linear, non-sequential manner. Hypertext links are created using a special software language known as HyperText Mark-Up Language (HTML).

Once created, Web pages reside on the Web, on Web servers or Web sites. A Web site can contain numerous Web pages. Web client machines running Web browsers can access these Web pages at Web sites via a communications protocol known as HyperText Transport Protocol (HTTP). Web browsers are software interfaces that run on World Wide Web clients to allow access to Web sites via a simple user interface. A Web browser allows a Web client to request a particular Web page from a Web site by specifying a Uniform Resource Locator (URL). A URL is a Web address that identifies the Web page and its location on the Web. When the appropriate Web site receives the URL, the Web page corresponding to the requested URL is located, and if required, HTML output is generated. The HTML output is then sent via HTTP to the client for formatting on the client's screen.

Although Web pages and Web sites are extremely simple to create, the proliferation of Web sites on the Internet highlighted a number of problems. The scope and ability of a Web page designer to change the content of the Web page was limited by the static nature of Web pages. Once created, a Web page remained static until it was manually modified. This in turn limited the ability of Web site managers to effectively manage their Web sites.

The Common Gateway Interface (CGI) standard was developed to resolve the problem of allowing dynamic content to be included in Web pages. CGI "calls" or procedures enable applications to generate dynamically created HTML output, thus creating Web pages with dynamic content. Once created, these CGI applications do not have to be modified in order to retrieve "new" or dynamic data. Instead, when the Web page is invoked, CGI "calls" or procedures are used to dynamically retrieve the necessary data and to generate a Web page.

CGI applications also enhanced the ability of Web site administrators to manage Web sites. Administrators no longer have to constantly update static Web pages. A number of vendors have developed tools for CGI based development, to address the issue of dynamic Web page generation. Companies like Spider™ and Bluestone™, for example, have each created development tools for CGI-based Web page development. Another company, Haht Software™, has developed a Web page generation tool that uses a BASIC-like scripting language, instead of a CGI scripting language.

Tools that generate CGI applications do not, however, resolve the problem of managing numerous Web pages and requests at a Web site. For example, a single company may maintain hundreds of Web pages at their Web site. Current Web server architecture also does not allow the Web server to efficiently manage the Web page and process Web client requests. Managing these hundreds of Web pages in a coherent manner and processing all requests for access to the Web pages is thus a difficult task. Existing development tools are limited in their capabilities to facilitate dynamic Web page generation, and do not address the issue of managing Web requests or Web sites.

SUMMARY OF THE INVENTION

It is therefore an object of the present invention to provide a method and apparatus for creating and managing custom Web sites. Specifically, the present invention claims a method and apparatus for managing dynamic web page generation requests.

In one embodiment, the present invention claims a computer-implemented method for managing a dynamic Web page generation request to a Web server, the computer-implemented method comprising the steps of routing the request from the Web server to a page server, the page server receiving the request and releasing the Web server to process other requests, processing the request, the processing being performed by the page server concurrently with the Web server, as the Web server processes the other requests, and dynamically generating a Web page in response to the request, the Web page including data dynamically retrieved from one or more data sources. Other embodiments also include connection caches to the one or more data sources, page caches for each page server, and custom HTML extension templates for configuring the Web page.

Other objects, features and advantages of the present invention will be apparent from the accompanying drawings and from the detailed description.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** illustrates a typical computer system in which the present invention operates.

FIG. **2** illustrates a typical prior art Web server environment.

FIG. **3** illustrates a typical prior art Web server environment in the form of a flow diagram.

FIG. **4** illustrates one embodiment of the presently claimed invention.

FIG. **5** illustrates the processing of a Web browser request in the farm of a flow diagram, according to one embodiment of the presently claimed invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The present invention relates to a method and apparatus for creating and managing custom Web sites. In the following detailed description, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent to one of ordinary skill in the art, however, that these specific details need not be used to practice the present invention. In other instances, well-known structures, interfaces and processes have not been shown in detail in order not to unnecessarily obscure the present invention.

FIG. **1** illustrates a typical computer system **100** in which the present invention operates. The preferred embodiment of

the present invention is implemented on an IBM™ Personal Computer manufactured by IBM Corporation of Armonk, N.Y. An alternate embodiment may be implemented on an RS/6000™ Workstation manufactured by IBM Corporation of Armonk, N.Y. It will be apparent to those of ordinary skill in the art that other computer system architectures may also be employed.

In general, such computer systems as illustrated by FIG. **1** comprise a bus **101** for communicating information, a processor **102** coupled with the bus **101** for processing information, main memory **103** coupled with the bus **101** for storing information and instructions for the processor **102**, a read-only memory **104** coupled with the bus **101** for storing static information and instructions for the processor **102**, a display device **105** coupled with the bus **101** for displaying information for a computer user, an input device **106** coupled with the bus **101** for communicating information and command selections to the processor **102**, and a mass storage device **107**, such as a magnetic disk and associated disk drive, coupled with the bus **101** for storing information and instructions. A data storage medium **108** containing digital information is configured to operate with mass storage device **107** to allow processor **102** access to the digital information on data storage medium **108** via bus **101**.

Processor **102** may be any of a wide variety of general purpose processors or microprocessors such as the Pentium™ microprocessor manufactured by Intel™ Corporation or the RS/6000™ processor manufactured by IBM Corporation. It will be apparent to those of ordinary skill in the art, however, that other varieties of processors may also be used in a particular computer system. Display device **105** may be a liquid crystal device, cathode ray tube (CRT), or other suitable display device. Mass storage device **107** may be a conventional hard disk drive, floppy disk drive, CD-ROM drive, or other magnetic or optical data storage device for reading and writing information stored on a hard disk, a floppy disk, a CD-ROM a magnetic tape, or other magnetic or optical data storage medium. Data storage medium **108** may be a hard disk, a floppy disk, a CD-ROM, a magnetic tape, or other magnetic or optical data storage medium.

In general, processor **102** retrieves processing instructions and data from a data storage medium **108** using mass storage device **107** and downloads this information into random access memory **103** for execution. Processor **102**, then executes an instruction stream from random access memory **103** or read-only memory **104**. Command selections and information input at input device **106** are used to direct the flow of instructions executed by processor **102**. Equivalent input device **106** may also be a pointing device such as a conventional mouse or trackball device. The results of this processing execution are then displayed on display device **105**.

The preferred embodiment of the present invention is implemented as a software module, which may be executed on a computer system such as computer system **100** in a conventional manner. Using well known techniques, the application software of the preferred embodiment is stored on data storage medium **108** and subsequently loaded into and executed within computer system **100**. Once initiated, the software of the preferred embodiment operates in the manner described below.

FIG. **2** illustrates a typical prior art Web server environment. Web client **200** can make URL requests to Web server **201** or Web server **202**. Web servers **201** and **202** include Web server executables, **201**(E) and **202**(E) respectively, that perform the processing of Web client requests. Each Web server may have a number of Web pages **201**(1)–(n) and **202**(1)–(n). Depending on the URL specified by the Web client **200**, the request may be routed by either Web server executable **201**(E) to Web page **201** (1), for example, or from Web server executable **202**(E) to Web page **202** (1). Web client **200** can continue making URL requests to retrieve other Web pages. Web client **200** can also use hyperlinks within each Web page to "jump" to other Web pages or to other locations within the same Web page.

FIG. **3** illustrates this prior art Web server environment in the form of a flow diagram. In processing block **300**, the Web client makes a URL request. This URL request is examined by the Web browser to determine the appropriate Web server to route the request to in processing block **302**. In processing block **304** the request is then transmitted from the Web browser to the appropriate Web server, and in processing block **306** the Web server executable examines the URL to determine whether it is a HTML document or a CGI application. If the request is for an HTML document **308**, then the Web server executable locates the document in processing block **310**. The document is then transmitted back through the requesting Web browser for formatting and display in processing block **312**.

If the URL request is for a CGI application **314**, however, the Web server executable locates the CGI application in processing block **316**. The CGI application then executes and outputs HTML output in processing block **318** and finally, the HTML output is transmitted back to requesting Web browser for formatting and display in processing block **320**.

This prior art Web server environment does not, however, provide any mechanism for managing the Web requests or the Web sites. As Web sites grow, and as the number of Web clients and requests increase, Web site management becomes a crucial need.

For example, a large Web site may receive thousands of requests or "hits" in a single day. Current Web servers process each of these requests on a single machine, namely the Web server machine. Although these machines may be running "multi-threaded" operating systems that allow transactions to be processed by independent "threads," all the threads are nevertheless on a single machine, sharing a processor. As such, the Web executable thread may hand off a request to a processing thread, but both threads will still have to be handled by the processor on the Web server machine. When numerous requests are being simultaneously processed by multiple threads on a single machine, the Web server can slow down significantly and become highly inefficient. The claimed invention addresses this need by utilizing a partitioned architecture to facilitate the creation and management of custom Web sites and servers.

FIG. **4** illustrates one embodiment of the presently claimed invention. Web client **200** issues a URL request that is processed to determined proper routing. In this embodiment, the request is routed to Web server **201**. Instead of Web server executable **201**(E) processing the URL request, however, Interceptor **400** intercepts the request and routes it to Dispatcher **402**. In one embodiment, Interceptor **400** resides on the Web server machine as an extension to Web server **201**. This embodiment is appropriate for Web servers such as Netsite™ from Netscape, that support such extensions. A number of public domain Web servers, such as NCSA™ from the National Center for Supercomputing Applications at the University of Illinois, Urbana-Champaign, however, do not provide support for

this type of extension. Thus, in an alternate embodiment, Interceptor **400** is an independent module, connected via an "intermediate program" to Web server **201**. This intermediate program can be a simple CGI application program that connects Interceptor **400** to Web server **201**. Alternate intermediate programs the perform the same functionality can also be implemented.

In one embodiment of the invention, Dispatcher **402** resides on a different machine than Web server **201**. This embodiment overcomes the limitation described above, in prior art Web servers, wherein all processing is performed by the processor on a single machine. By routing the request to Dispatcher **402** residing on a different machine than the Web server executable **201**(E), the request can then be processed by a different processor than the Web server executable **201**(E). Web server executable **201**(E) is thus free to continue servicing client requests on Web server **201** while the request is processed "off-line," at the machine on which Dispatcher **402** resides.

Dispatcher **402** can, however, also reside on the same machine as the Web server. The Web site administrator has the option of configuring Dispatcher **402** on the same machine as Web server **201**, taking into account a variety of factors pertinent to a particular Web site, such as the size of the Web site, the number of Web pages and the number of hits at the Web site. Although this embodiment will not enjoy the advantage described above, namely off-loading the processing of Web requests from the Web server machine, the embodiment does allow flexibility for a small Web site to grow. For example, a small Web site administrator can use a single machine for both Dispatcher **402** and Web server **201** initially, then off-load Dispatcher **402** onto a separate machine as the Web site grows. The Web site can thus take advantage of other features of the present invention regardless of whether the site has separate machines configured as Web servers and dispatchers.

Dispatcher **402** receives the intercepted request and then dispatches the request to one of a number of Page servers **404** (1)–(n). For example, if Page server **404** (1) receives the dispatched request, it processes the request and retrieves the data from an appropriate data source, such as data source **406**, data source **408**, or data source **410**. Data sources, as used in the present application, include databases, spreadsheets, files and any other type of data repository. Page server **404** (1) can retrieve data from more than one data source and incorporate the data from these multiple data sources in a single Web page.

In one embodiment, each Page server **404**(1)–(n) resides on a separate machine on the network to distribute the processing of the request. Dispatcher **402** maintains a variety of information regarding each Page server on the network, and dispatches requests based on this information. For example, Dispatcher **402** retains dynamic information regarding the data sources that any given Page server can access. Dispatcher **402** thus examines a particular request and determines which Page servers can service the URL request. Dispatcher **402** then hands off the request to the appropriate Page server.

For example, if the URL request requires financial data from data source **408**, dispatcher **402** will first examine an information list. Dispatcher **402** may determine that Page server **404**(3), for example, has access to the requisite data in data source **408**. Dispatcher **402** will thus route the URL request to Page server **404**(3). This "connection caching" functionality is described in more detail below, under the heading "Performance."

Alternately, Dispatcher **402** also has the ability to determine whether a particular Page server already has the necessary data cached in the Page server's page cache (described in more detail below, under the heading "Performance"). Dispatcher **402** may thus determine that Page server **404**(1) and **404**(2) are both logged into Data source **408**, but that Page server **404**(2) has the financial information already cached in Page server **404**(2)'s page cache. In this case, Dispatcher **402** will route the URL request to Page server **404**(2) to more efficiently process the request.

Finally, Dispatcher **402** may determine that a number or all Page servers **404**(1)–(n) are logged into Data source **408**. In this scenario, Dispatcher **402** can examine the number of requests that each Page server is servicing and route the request to the least busy page server. This "load balancing" capability can significantly increase performance at a busy Web site and is discussed in more detail below, under the heading "Scalability".

If, for example, Page server **404**(2), receives the request, Page server **404**(2) will process the request. While Page server **404**(2) is processing the request, Web server executable **201**(E) can concurrently process other Web client requests. This partitioned architecture thus allows both Page server **404**(2) and Web server executable **201**(E) to simultaneously process different requests, thus increasing the efficiency of the Web site. Page server **404**(2) dynamically generates a Web page in response to the Web client request, and the dynamic Web page is then either transmitted back to requesting Web client **200** or stored on a machine that is accessible to Web server **201**, for later retrieval.

One embodiment of the claimed invention also provides a Web page designer with HTML extensions, or "dyna" tags. These dyna tags provide customized HTML functionality to a Web page designer, to allow the designer to build customized HTML templates that specify the source and placement of retrieved data. For example, in one embodiment, a "dynatext" HTML extension tag specifies a data source and a column name to allow the HTML template to identify the data source to log into and the column name from which to retrieve data. Alternatively, "dyna-anchor" tags allow the designer to build hyperlink queries while "dynablock" tags provide the designer with the ability to iterate through blocks of data. Page servers use these HTML templates to create dynamic Web pages. Then, as described above, these dynamic Web pages are either transmitted back to requesting Web client **200** or stored on a machine that is accessible to Web server **201**, for later retrieval.

The presently claimed invention provides numerous advantages over prior art Web servers, including advantages in the areas of performance, security, extensibility and scalability.

Performance

One embodiment of the claimed invention utilizes connection caching and page caching to improve performance. Each Page server can be configured to maintain a cache of connections to numerous data sources. For example, as illustrated in FIG. **4**, Page server **404**(1) can retrieve data from data source **406**, data source **408** or data source **410**. Page server **404**(1) can maintain connection cache **412**(1), containing connections to each of data source **406**, data source **408** and data source **410**, thus eliminating connect times from the Page servers to those data sources.

Additionally, another embodiment of the present invention supports the caching of finished Web pages, to optimize

the performance of the data source being utilized. This "page caching" feature, illustrated in FIG. **4** as Page cache **414**, allows the Web site administrator to optimize the performance of data sources by caching Web pages that are repeatedly accessed. Once the Web page is cached, subsequent requests or "hits" will utilize the cached Web page rather than re-accessing the data source. This can radically improve the performance of the data source.

Security

The present invention allows the Web site administrator to utilize multiple levels of security to manage the Web site. In one embodiment, the Page server can utilize all standard encryption and site security features provided by the Web server. In another embodiment, the Page server can be configured to bypass connection caches **412**(1)–(n), described above, for a particular data source and to require entry of a user-supplied identification and password for the particular data source the user is trying to access.

Additionally, another embodiment of the presently claimed invention requires no real-time access of data sources. The Web page caching ability, described above, enables additional security for those sites that want to publish non-interactive content from internal information systems, but do not want real-time Internet accessibility to those internal information systems. In this instance, the Page server can act as a "replication and staging gent" and create Web pages in batches, rather than in real-time. These "replicated" Web pages are then "staged" for access at a later time, and access o the Web pages in this scenario is possible even if the Page server and dispatcher are not present later.

In yet another embodiment, the Page server can make a single pass through a Web library, and compile a Web site that exists in the traditional form of separately available files. A Web library is a collection of related Web books and Web pages. More specifically, the Web library is a hierarchical organization of Web document templates, together with all the associated data source information. Information about an entire Web site is thus contained in a single physical file, thus simplifying the problem of deploying Web sites across multiple Page servers. The process of deploying the Web site in this embodiment is essentially a simple copy of a single file.

Extensibility

One embodiment of the present invention provides the Web site administrator with Object Linking and Embedding (OLE) 2.0 extensions to extend the page creation process. These OLE 2.0 extensions also allow information submitted over the Web to be processed with user-supplied functionality. Utilizing development tools such as Visual Basic, Visual C++ or PowerBuilder that support the creation of OLE 2.0 automation, the Web site administrator can add features and modify the behavior of the Page servers described above. This extensibility allows one embodiment of the claimed invention to be incorporated with existing technology to develop an infinite number of custom web servers.

For example, OLE 2.0 extensions allow a Web site administrator to encapsulate existing business rules in an OLE 2.0 automation interface, to be accessed over the Web. One example of a business rule is the steps involved in the payoff on an installment or mortgage loan. The payoff may involve, for example, taking into account the current balance, the date and the interest accrued since the last payment. Most organizations already have this type of business rule implemented using various applications, such as Visual Basic for client-server environments, or CICS programs on mainframes. If these applications are OLE 2.0 compliant, the Page server "dynaobject" HTML extension tag can be used to encapsulated the application in an OLE 2.0 automation interface. The Page server is thus extensible, and can incorporate the existing application with the new Page server functionality.

Scalability

One embodiment of the claimed invention allows "plug and play" scalability. As described above, referring to FIG. **4**, Dispatcher **402** maintains information about all the Page servers configured to be serviced by Dispatcher **402**. Any number of Page servers can thus be "plugged" into the configuration illustrated in FIG. **4**, and the Page servers will be instantly activated as the information is dynamically updated in Dispatcher **402**. The Web site administrator can thus manage the overhead of each Page server and modify each Page server's load, as necessary, to improve performance. In this manner, each Page server will cooperate with other Page servers within a multi-server environment. Dispatcher **402** can examine the load on each Page server and route new requests according to each Page server's available resources. This "load-balancing" across multiple Page servers can significantly increase a Web site's performance.

FIG. **5** illustrates the processing of a Web browser request in the form of a flow diagram, according to one embodiment of the presently claimed invention. A Web browser sends a URL request to a Web server in processing block **500**. In processing block **502**, the Web server receives the URL request, and an interceptor then intercepts the handling of the request in processing block **504**. The interceptor connects to a dispatcher and sends the URL request to the dispatcher in processing block **506**. In processing block **508**, the dispatcher determines which Page servers can handle the request. The dispatcher also determines which Page server is processing the fewest requests in processing block **510**, and in processing block **512**, the dispatcher sends the URL request to an appropriate Page server. The Page server receives the request and produces an HTML document in processing block **514**. The Page server then responds to the dispatcher with notification of the name of the cached HTML document in processing block **516**. In processing block **518**, the dispatcher responds to the interceptor with the document name, and the interceptor then replaces the requested URL with the newly generated HTML document in processing block **520**. The Web server then sends the new HTML document to the requesting client in processing block **522**. Finally, the Web browser receives and displays the HTML document created by the Page server at processing block **524**.

Thus, a method and apparatus for creating and managing custom Web sites is disclosed. These specific arrangements and methods described herein are merely illustrative of the principles of the present invention. Numerous modifications in form and detail may be made by those of ordinary skill in the art without departing from the scope of the present invention. Although this invention has been shown in relation to a particular preferred embodiment, it should not be considered so limited. Rather, the present invention is limited only by the scope of the appended claims.

We claim:

1. A computer-implemented method for managing a dynamic Web page generation request to a Web server, said computer-implemented method comprising the steps of:

routing said request from said Web server to a page server, said page server receiving said request and releasing

said Web server to process other requests, wherein said routing step further includes the steps of intercepting said request at said Web server, routing said request from said Web server to a dispatcher, and dispatching said request to said page server;

processing said request, said processing being performed by said page server while said Web server concurrently processes said other requests; and

dynamically generating a Web page in response to said request, said Web page including data dynamically retrieved from one or more data sources.

2. The computer-implemented method in claim **1** wherein said step of processing said request includes the step of identifying said one or more data sources from which to retrieve said data.

3. The computer-implemented method in claim **2** wherein said step of dynamically generating said Web page includes the step of dynamically retrieving said data from said one or more data sources.

4. The computer-implemented method in claim **3** wherein said step of processing said request includes the step of said page server maintaining a connection cache to said one or more data sources.

5. The computer-implemented method in claim **3** wherein said step of processing said request includes the step of logging into said one or more data sources.

6. The computer-implemented method in claim **3** wherein said step of dynamically generating said Web page includes the step of maintaining a page cache containing said Web page.

7. The computer-implemented method in claim **3** wherein said page server includes custom HTML extension templates for configuring said Web page.

8. The computer-implemented method in claim **7** wherein said step of processing said request further includes the step of inserting said dynamically retrieved data from said one or more data sources into said custom HTML extension templates.

9. A networked system for managing a dynamic Web page generation request, said system comprising:

one or more data sources;

a page server having a processing means;

a first computer system including means for generating said request; and

a second computer system including means for receiving said request from said first computer, said second computer system also including a router, said router routing said request from said second computer system to said page server, wherein said routing further includes intercepting said request at said second computer, routing said request from said second computer to a dispatcher, and dispatching said request to said page server said page server receiving said request and releasing said second computer system to process other requests, said page server processing means processing said request and dynamically generating a Web page in response to said request, said Web page including data dynamically retrieved from said one or more data sources.

10. The networked system in claim **9** wherein said router in said second computer system includes:

an interceptor intercepting said request at said second computer system and routing said request; and

a dispatcher receiving said routed request from said interceptor and dispatching said request to said page server.

11. A machine readable medium having stored thereon data representing sequences of instructions, which when executed by a computer system, cause said computer system to perform the steps of:

routing a dynamic Web page generation request from a Web server to a page server, said page server receiving said request and releasing said Web server to process other requests wherein said routing step further includes the steps of intercepting said request at said Web server, routing said request from said Web server to a dispatcher, and dispatching said request to said page server;

processing said request, said processing being performed by said page server while said Web server concurrently processes said other requests; and

dynamically generating a Web page, said Web page including data retrieved from one or more data sources.

* * * * *

A2

US006415335B1

(12) **United States Patent**
Lowery et al.

(10) **Patent No.: US 6,415,335 B1**
(45) **Date of Patent: *Jul. 2, 2002**

(54) **SYSTEM AND METHOD FOR MANAGING DYNAMIC WEB PAGE GENERATION REQUESTS**

(75) Inventors: **Keith Lowery**, Richardson; **Andrew B. Levine**, Plano; **Ronald L. Howell**, Rowlett, all of TX (US)

(73) Assignee: **epicRealm Operating Inc.**, Richardson, TX (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **09/234,048**

(22) Filed: **Jan. 19, 1999**

Related U.S. Application Data

(62) Division of application No. 08/636,477, filed on Apr. 23, 1996, now Pat. No. 5,894,554.

(51) **Int. Cl.**[7] **G06F 13/14**; G06F 13/20
(52) **U.S. Cl.** **710/5**; 710/7; 709/219; 709/223; 709/238
(58) **Field of Search** 709/238, 223, 709/219; 710/5, 7, 20–21

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,866,706	A		9/1989	Christophersen et al. .. 370/85.7
5,341,499	A		8/1994	Doragh 395/700
5,392,400	A		2/1995	Berkowitz et al. 395/200
5,404,522	A		4/1995	Carmon et al. 395/650
5,404,523	A		4/1995	DellaFera et al. 395/650
5,404,527	A	*	4/1995	Irwin et al. 395/700
5,452,460	A		9/1995	Distelberg et al. 395/700
5,532,838	A		7/1996	Barbari 358/400
5,701,463	A	*	12/1997	Malcolm 395/610
5,751,956	A		5/1998	Kirsch 395/200.33
5,752,246	A	*	5/1998	Rogers et al. 707/10
5,754,772	A	*	5/1998	Leaf 395/200.33
5,761,673	A	*	6/1998	Bookman et al. 707/104
5,774,660	A		6/1998	Brendel et al. 395/200.31
5,774,668	A		6/1998	Choquier et al. 395/200.53

OTHER PUBLICATIONS

Hoffner, 'Inter–operability and distributed application platform design', Web URL:http://www.ansa.co.uk/, 1995, pp. 342–356.*

Mourad et al, 'Scalable Web Server Architectures', IEEE, Jun. 1997, pp. 12–16.*

Dias et al, 'A Scalable and Highly Available Web Server', IEEE, 1996, pp. 85–92.*

'Single System Image and Load Balancing for Network Access to a Loosely Coupled Complex', IBM TDB, vol. 34, Feb. 1992, pp. 464–467.*

Dias, Daniel M., et al.; A Scalable and Highly Available Web Server; IBM Research Division; T.J. Watson Research Center; 7 pages.

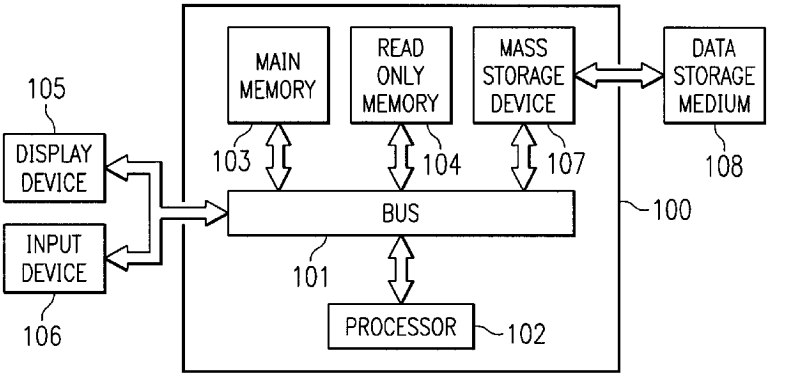
(List continued on next page.)

Primary Examiner—Jeffrey Gaffin
Assistant Examiner—Rehana Perveen
(74) *Attorney, Agent, or Firm*—Baker Botts L.L.P.

(57) **ABSTRACT**

The present invention teaches a method and apparatus for creating and managing custom Web sites. Specifically, one embodiment of the present invention claims a computer-implemented method for managing a dynamic Web page generation request to a Web server, the computer-implemented method comprising the steps of routing the request from the Web server to a page server, the page server receiving the request and releasing the Web server to process other requests, processing the request, the processing being performed by the page server concurrently with the Web server, as the Web server processes the other requests, and dynamically generating a Web page in response to the request, the Web page including data dynamically retrieved from one or more data sources.

29 Claims, 4 Drawing Sheets



OTHER PUBLICATIONS

Andresen, Daniel, Et Al.; Scalability Issues for High Performance Digital Libraries on the World Wide Web; Department of Computer Science; University of California at Santa Barbara; 10 pages.

Andresen, Daniel, Et Al.; SWEB: Towards a Scalable World Wide Web Server on Multicomputers; Department of Computer Science; University of California at Santa Barbara; 7 pages.

Holmedahl, Vegard; Et Al.; Cooperative Caching of Dynamic Content on a Distributed Web Server; Department of Computer Science; University of California at Santa Barbara; 8 pages.

Overson, Nicole; NeXT Ships WebObjects—On Time—As Promised; Deja.com: NeXT Ships WebObjects—On Time—As Promishttp://X28..deja.com/=dnc/ST_m=ps...EXT=927585438. 1744765032&hitnum=33.

International Search Report; 7 pages; dated Aug. 21, 1997.

Birman, Kenneth P. and van Renesse, Robbert; Software for Reliable Networks; Scientific American; May 1996; pp. 64–69.

"Beyond the Web: Excavating the Real World Via Mosaic"; Goldberg et al.; Second International WWW Conference; Oct. 17, 1994.

* cited by examiner

FIG. 1

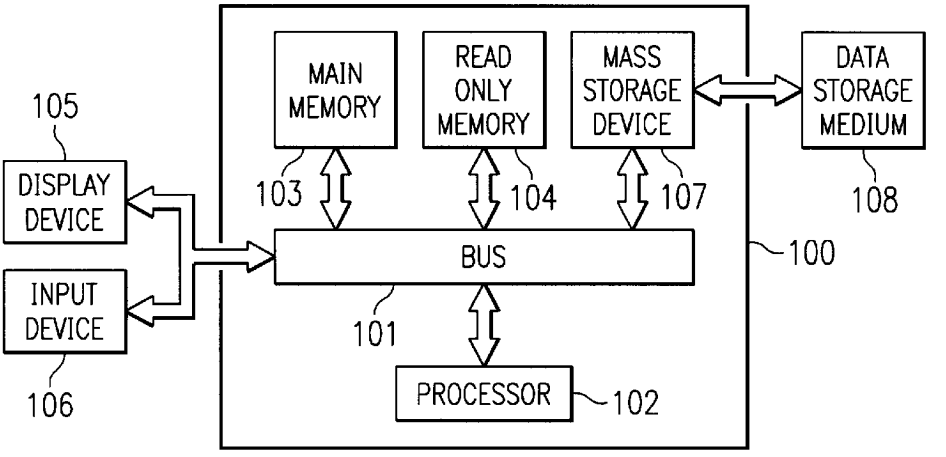


FIG. 2
(PRIOR ART)

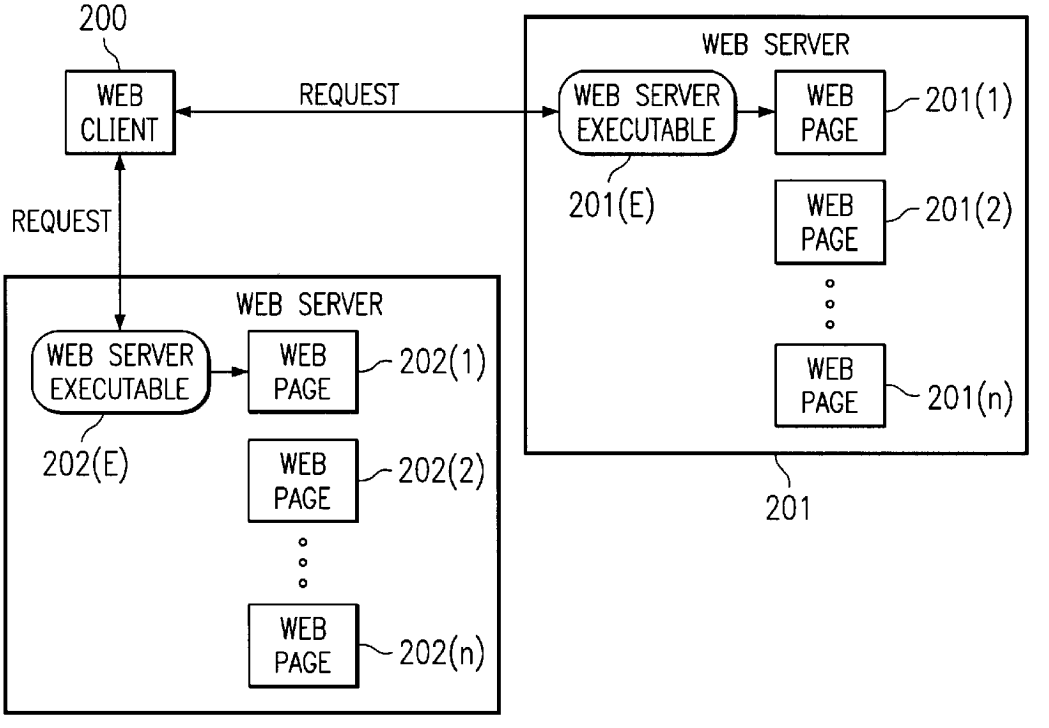
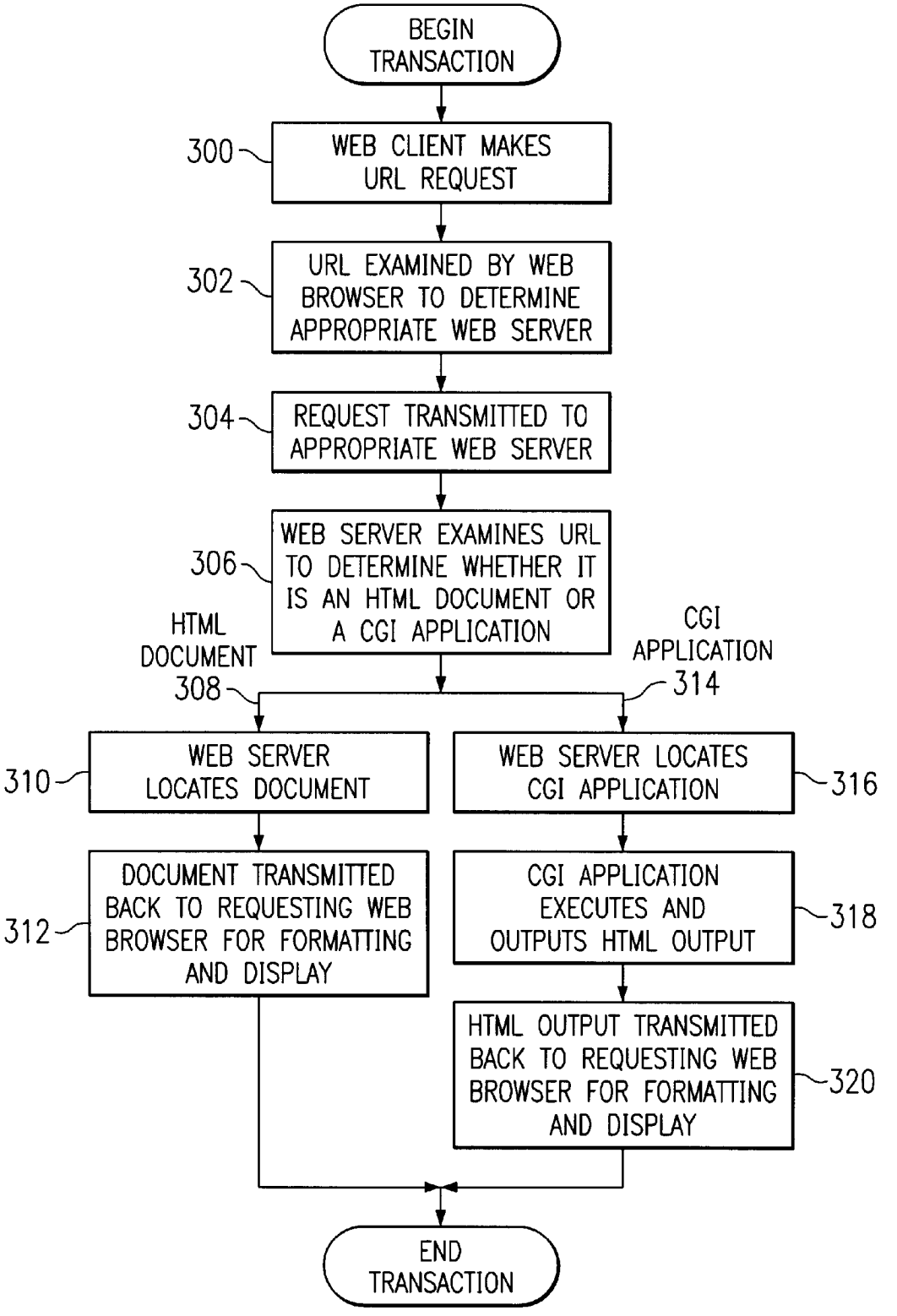


FIG. 3
(PRIOR ART)



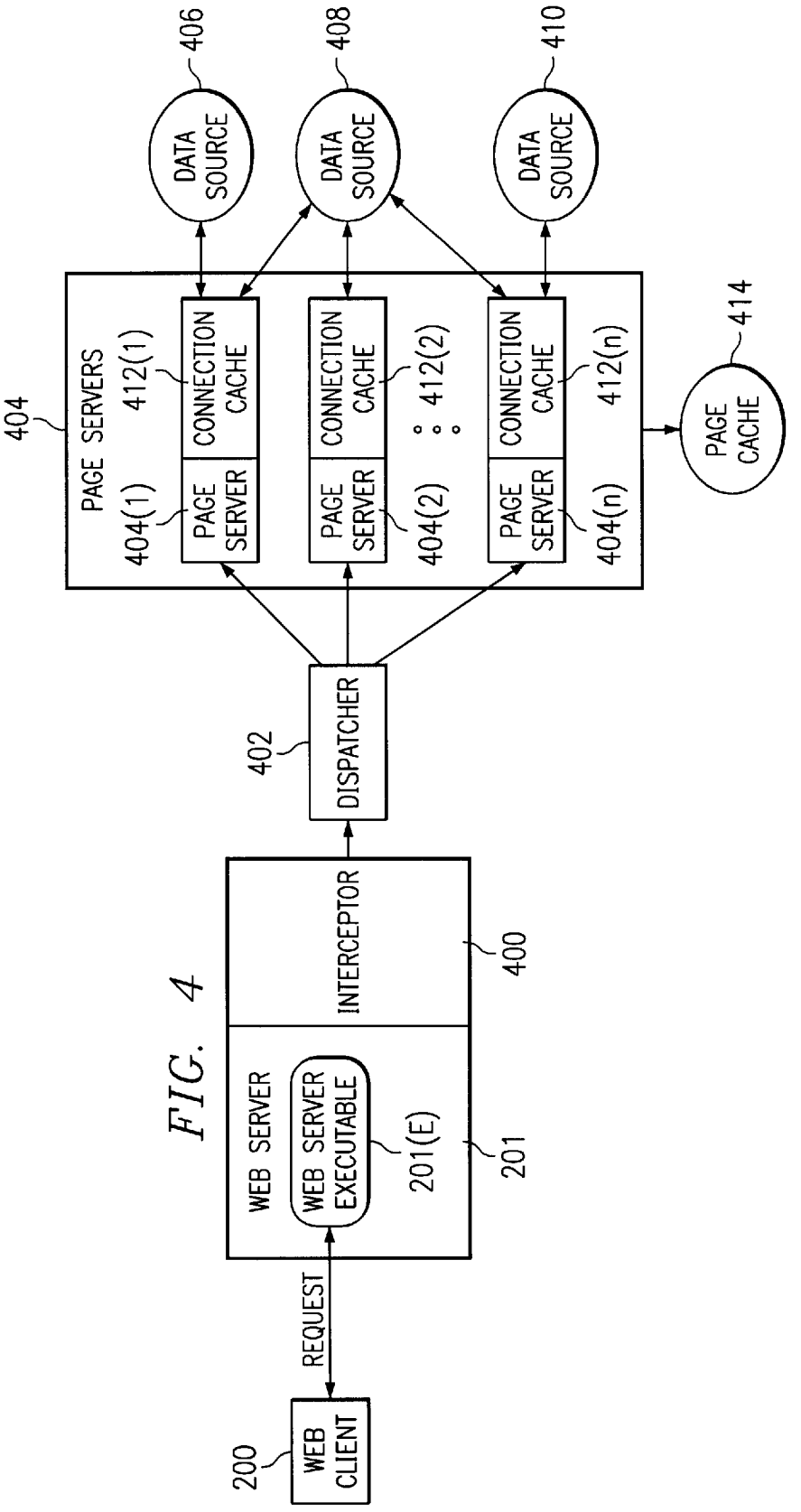
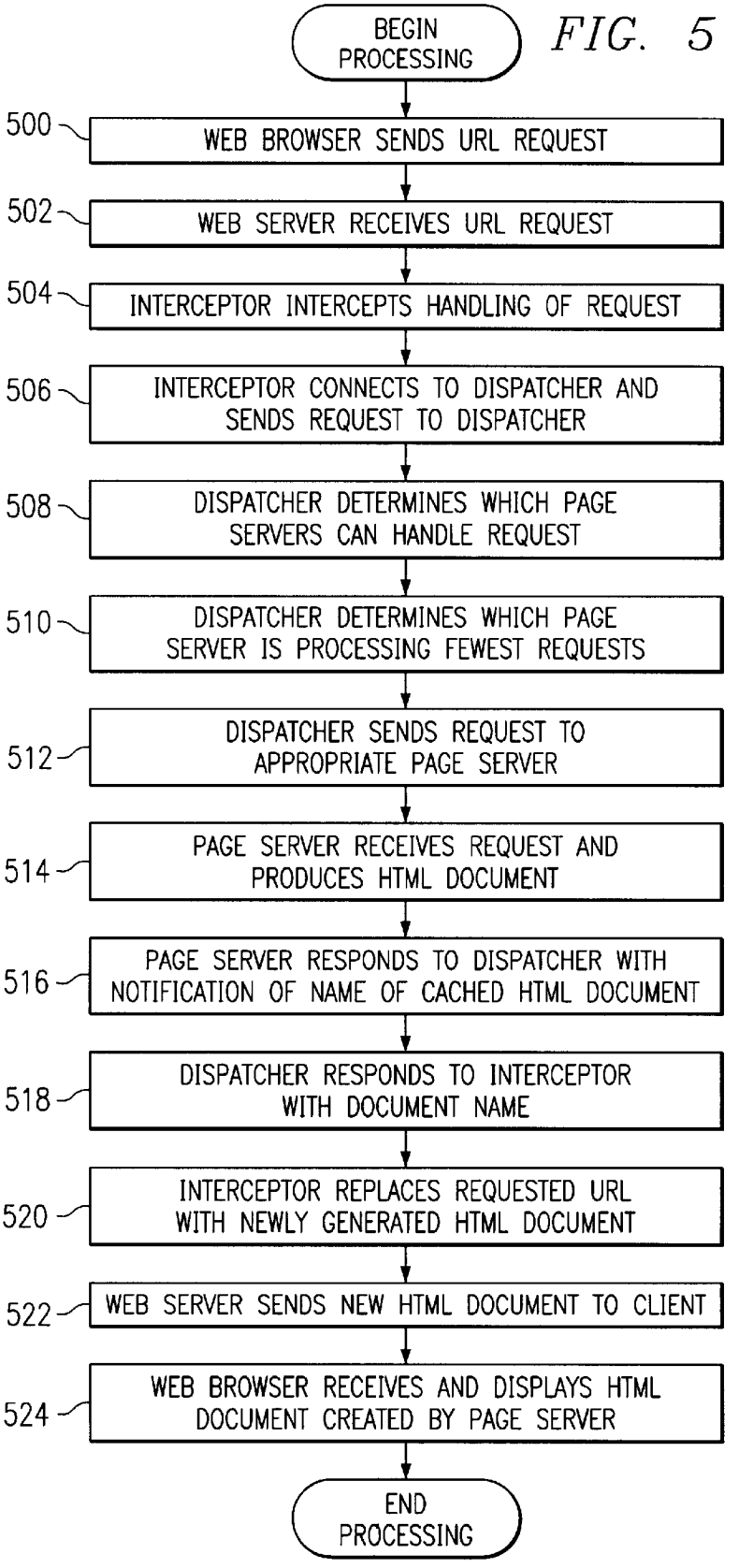

406
408
410
DATA SOURCE
DATA SOURCE
DATA SOURCE
404
PAGE SERVERS
412(1)
CONNECTION CACHE
PAGE SERVER
404(1)
412(2)
404(2)
412(n)
404(n)
414
PAGE CACHE
402
DISPATCHER
INTERCEPTOR
400
WEB SERVER
WEB SERVER EXECUTABLE
201(E)
201
REQUEST
200
WEB CLIENT

FIG. 4

FIG. 5


BEGIN PROCESSING
500
WEB BROWSER SENDS URL REQUEST
502
WEB SERVER RECEIVES URL REQUEST
504
INTERCEPTOR INTERCEPTS HANDLING OF REQUEST
506
INTERCEPTOR CONNECTS TO DISPATCHER AND SENDS REQUEST TO DISPATCHER
508
DISPATCHER DETERMINES WHICH PAGE SERVERS CAN HANDLE REQUEST
510
DISPATCHER DETERMINES WHICH PAGE SERVER IS PROCESSING FEWEST REQUESTS
512
DISPATCHER SENDS REQUEST TO APPROPRIATE PAGE SERVER
514
PAGE SERVER RECEIVES REQUEST AND PRODUCES HTML DOCUMENT
516
PAGE SERVER RESPONDS TO DISPATCHER WITH NOTIFICATION OF NAME OF CACHED HTML DOCUMENT
518
DISPATCHER RESPONDS TO INTERCEPTOR WITH DOCUMENT NAME
520
INTERCEPTOR REPLACES REQUESTED URL WITH NEWLY GENERATED HTML DOCUMENT
522
WEB SERVER SENDS NEW HTML DOCUMENT TO CLIENT
524
WEB BROWSER RECEIVES AND DISPLAYS HTML DOCUMENT CREATED BY PAGE SERVER
END PROCESSING

SYSTEM AND METHOD FOR MANAGING DYNAMIC WEB PAGE GENERATION REQUESTS

This application is a division of Ser. No. 08/636,477, filed Apr. 23, 1996, now U.S. Pat. No. 5,894,554.

FIELD OF THE INVENTION

The present invention relates to the field of Internet technology. Specifically, the present invention relates to the creation and management of custom World Wide Web sites.

DESCRIPTION OF RELATED ART

The World Wide Web (the Web) represents all of the computers on the Internet that offer users access to information on the Internet via interactive documents or Web pages. These Web pages contain hypertext links that are used to connect any combination of graphics, audio, video and text, in a non-linear, non-sequential manner. Hypertext links are created using a special software language known as HyperText Mark-Up Language (HTML).

Once created, Web pages reside on the Web, on Web servers or Web sites. A Web site can contain numerous Web pages. Web client machines running Web browsers can access these Web pages at Web sites via a communications protocol known as HyperText Transport Protocol (HTTP). Web browsers are software interfaces that run on World Wide Web clients to allow access to Web sites via a simple user interface. A Web browser allows a Web client to request a particular Web page from a Web site by specifying a Uniform Resource Locator (URL). A URL is a Web address that identifies the Web page and its location on the Web. When the appropriate Web site receives the URL, the Web page corresponding to the requested URL is located, and if required, HTML output is generated. The HTML output is then sent via HTTP to the client for formatting on the client's screen.

Although Web pages and Web sites are extremely simple to create, the proliferation of Web sites on the Internet highlighted a number of problems. The scope and ability of a Web page designer to change the content of the Web page was limited by the static nature of Web pages. Once created, a Web page remained static until it was manually modified. This in turn limited the ability of Web site managers to effectively manage their Web sites.

The Common Gateway Interface (CGI) standard was developed to resolve the problem of allowing dynamic content to be included in Web pages. CGI "calls" or procedures enable applications to generate dynamically created HTML output, thus creating Web pages with dynamic content. Once created, these CGI applications do not have to be modified in order to retrieve "new" or dynamic data. Instead, when the Web page is invoked, CGI "calls" or procedures are used to dynamically retrieve the necessary data and to generate a Web page.

CGI applications also enhanced the ability of Web site administrators to manage Web sites. Administrators no longer have to constantly update static Web pages. A number of vendors have developed tools for CGI based development, to address the issue of dynamic Web page generation. Companies like Spider™ and Bluestone™, for example, have each created development tools for CGI-based Web page development. Another company, Haht Software™, has developed a Web page generation tool that uses a BASIC-like scripting language, instead of a CGI scripting language.

Tools that generate CGI applications do not, however, resolve the problem of managing numerous Web pages and requests at a Web site. For example, a single company may maintain hundreds of Web pages at their Web site. Current Web server architecture also does not allow the Web server to efficiently manage the Web page and process Web client requests. Managing these hundreds of Web pages in a coherent manner and processing all requests for access to the Web pages is thus a difficult task. Existing development tools are limited in their capabilities to facilitate dynamic Web page generation, and do not address the issue of managing Web requests or Web sites.

SUMMARY OF THE INVENTION

It is therefore an object of the present invention to provide a method and apparatus for creating and managing custom Web sites. Specifically, the present invention claims a method and apparatus for managing dynamic web page generation requests.

In one embodiment, the present invention claims a computer-implemented method for managing a dynamic Web page generation request to a Web server, the computer-implemented method comprising the steps of routing the request from the Web server to a page server, the page server receiving the request and releasing the Web server to process other requests, processing the request, the processing being performed by the page server concurrently with the Web server, as the Web server processes the other requests, and dynamically generating a Web page in response to the request, the Web page including data dynamically retrieved from one or more data sources. Other embodiments also include connection caches to the one or more data sources, page caches for each page server, and custom HTML extension templates for configuring the Web page.

Other objects, features and advantages of the present invention will be apparent from the accompanying drawings and from the detailed description.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** illustrates a typical computer system in which the present invention operates.

FIG. **2** illustrates a typical prior art Web server environment.

FIG. **3** illustrates a typical prior art Web server environment in the form of a flow diagram.

FIG. **4** illustrates one embodiment of the presently claimed invention.

FIG. **5** illustrates the processing of a Web browser request in the form of a flow diagram, according to one embodiment of the presently claimed invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The present invention relates to a method and apparatus for creating and managing custom Web sites. In the following detailed description, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent to one of ordinary skill in the art, however, that these specific details need not be used to practice the present invention. In other instances, well-known structures, interfaces and processes have not been shown in detail in order not to unnecessarily obscure the present invention.

FIG. **1** illustrates a typical computer system **100** in which the present invention operates. The preferred embodiment of

the present invention is implemented on an IBM™ Personal Computer manufactured by IBM Corporation of Armonk, New York. An alternate embodiment may be implemented on an RS/6000™ Workstation manufactured by IBM Corporation of Armonk, New York. It will be apparent to those of ordinary skill in the art that other computer system architectures may also be employed.

In general, such computer systems as illustrated by FIG. **1** comprise a bus **101** for communicating information, a processor **102** coupled with the bus **101** for processing information, main memory **103** coupled with the bus **101** for storing information and instructions for the processor **102**, a read-only memory **104** coupled with the bus **101** for storing static information and instructions for the processor **102**, a display device **105** coupled with the bus **101** for displaying information for a computer user, an input device **106** coupled with the bus **101** for communicating information and command selections to the processor **102**, and a mass storage device **107**, such as a magnetic disk and associated disk drive, coupled with the bus **101** for storing information and instructions. A data storage medium **108** containing digital information is configured to operate with mass storage device **107** to allow processor **102** access to the digital information on data storage medium **108** via bus **101**.

Processor **102** may be any of a wide variety of general purpose processors or microprocessors such as the Pentium™ microprocessor manufactured by Intel™ Corporation or the RS/6000™ processor manufactured by IBM Corporation. It will be apparent to those of ordinary skill in the art, however, that other varieties of processors may also be used in a particular computer system. Display device **105** may be a liquid crystal device, cathode ray tube (CRT), or other suitable display device. Mass storage device **107** may be a conventional hard disk drive, floppy disk drive, CD-ROM drive, or other magnetic or optical data storage device for reading and writing information stored on a hard disk, a floppy disk, a CD-ROM a magnetic tape, or other magnetic or optical data storage medium. Data storage medium **108** may be a hard disk, a floppy disk, a CD-ROM, a magnetic tape, or other magnetic or optical data storage medium.

In general, processor **102** retrieves processing instructions and data from a data storage medium **108** using mass storage device **107** and downloads this information into random access memory **103** for execution. Processor **102**, then executes an instruction stream from random access memory **103** or read-only memory **104**. Command selections and information input at input device **106** are used to direct the flow of instructions executed by processor **102**. Equivalent input device **106** may also be a pointing device such as a conventional mouse or trackball device. The results of this processing execution are then displayed on display device **105**.

The preferred embodiment of the present invention is implemented as a software module, which may be executed on a computer system such as computer system **100** in a conventional manner. Using well known techniques, the application software of the preferred embodiment is stored on data storage medium **108** and subsequently loaded into and executed within computer system **100**. Once initiated, the software of the preferred embodiment operates in the manner described below.

FIG. **2** illustrates a typical prior art Web server environment. Web client **200** can make URL requests to Web server **201** or Web server **202**. Web servers **201** and **202** include Web server executables, **201**(E) and **202**(E) respectively, that perform the processing of Web client requests. Each Web server may have a number of Web pages **201(1)**–(n) and **202(1)**–(n). Depending on the URL specified by the Web client **200**, the request may be routed by either Web server executable **201**(E) to Web page **201 (1)**, for example, or from Web server executable **202**(E) to Web page **202 (1)**. Web client **200** can continue making URL requests to retrieve other Web pages. Web client **200** can also use hyperlinks within each Web page to "jump" to other Web pages or to other locations within the same Web page.

FIG. **3** illustrates this prior art Web server environment in the form of a flow diagram. In processing block **300**, the Web client makes a URL request. This URL request is examined by the Web browser to determine the appropriate Web server to route the request to in processing block **302**. In processing block **304** the request is then transmitted from the Web browser to the appropriate Web server, and in processing block **306** the Web server executable examines the URL to determine whether it is a HTML document or a CGI application. If the request is for an HTML document **308**, then the Web server executable locates the document in processing block **310**. The document is then transmitted back through the requesting Web browser for formatting and display in processing block **312**.

If the URL request is for a CGI application **314**, however, the Web server executable locates the CGI application in processing block **316**. The CGI application then executes and outputs HTML output in processing block **318** and finally, the HTML output is transmitted back to requesting Web browser for formatting and display in processing block **320**.

This prior art Web server environment does not, however, provide any mechanism for managing the Web requests or the Web sites. As Web sites grow, and as the number of Web clients and requests increase, Web site management becomes a crucial need.

For example, a large Web site may receive thousands of requests or "hits" in a single day. Current Web servers process each of these requests on a single machine, namely the Web server machine. Although these machines may be running "multi-threaded" operating systems that allow transactions to be processed by independent "threads," all the threads are nevertheless on a single machine, sharing a processor. As such, the Web executable thread may hand off a request to a processing thread, but both threads will still have to be handled by the processor on the Web server machine. When numerous requests are being simultaneously processed by multiple threads on a single machine, the Web server can slow down significantly and become highly inefficient. The claimed invention addresses this need by utilizing a partitioned architecture to facilitate the creation and management of custom Web sites and servers.

FIG. **4** illustrates one embodiment of the presently claimed invention. Web client **200** issues a URL request that is processed to determined proper routing. In this embodiment, the request is routed to Web server **201**. Instead of Web server executable **201**(E) processing the URL request, however, Interceptor **400** intercepts the request and routes it to Dispatcher **402**. In one embodiment, Interceptor **400** resides on the Web server machine as an extension to Web server **201**. This embodiment is appropriate for Web servers such as Netsite™ from Netscape, that support such extensions. A number of public domain Web servers, such as NCSA™ from the National Center for Supercomputing Applications at the University of Illinois, Urbana-Champaign, however, do not provide support for

this type of extension. Thus, in an alternate embodiment, Interceptor **400** is an independent module, connected via an "intermediate program" to Web server **201**. This intermediate program can be a simple CGI application program that connects Interceptor **400** to Web server **201**. Alternate intermediate programs the perform the same functionality can also be implemented.

In one embodiment of the invention, Dispatcher **402** resides on a different machine than Web server **201**. This embodiment overcomes the limitation described above, in prior art Web servers, wherein all processing is performed by the processor on a single machine. By routing the request to Dispatcher **402** residing on a different machine than the Web server executable **201**(E), the request can then be processed by a different processor than the Web server executable **201**(E). Web server executable **201**(E) is thus free to continue servicing client requests on Web server **201** while the request is processed "off-line," at the machine on which Dispatcher **402** resides.

Dispatcher **402** can, however, also reside on the same machine as the Web server. The Web site administrator has the option of configuring Dispatcher **402** on the same machine as Web server **201**, taking into account a variety of factors pertinent to a particular Web site, such as the size of the Web site, the number of Web pages and the number of hits at the Web site. Although this embodiment will not enjoy the advantage described above, namely off-loading the processing of Web requests from the Web server machine, the embodiment does allow flexibility for a small Web site to grow. For example, a small Web site administrator can use a single machine for both Dispatcher **402** and Web server **201** initially, then off-load Dispatcher **402** onto a separate machine as the Web site grows. The Web site can thus take advantage of other features of the present invention regardless of whether the site has separate machines configured as Web servers and dispatchers.

Dispatcher **402** receives the intercepted request and then dispatches the request to one of a number of Page servers **404 (1)**–(n). For example, if Page server **404 (1)** receives the dispatched request, it processes the request and retrieves the data from an appropriate data source, such as data source **406**, data source **408**, or data source **410**. Data sources, as used in the present application, include databases, spreadsheets, files and any other type of data repository. Page server **404 (1)** can retrieve data from more than one data source and incorporate the data from these multiple data sources in a single Web page.

In one embodiment, each Page server **404(1)**–(n) resides on a separate machine on the network to distribute the processing of the request. Dispatcher **402** maintains a variety of information regarding each Page server on the network, and dispatches requests based on this information. For example, Dispatcher **402** retains dynamic information regarding the data sources that any given Page server can access. Dispatcher **402** thus examines a particular request and determines which Page servers can service the URL request. Dispatcher **402** then hands off the request to the appropriate Page server.

For example, if the URL request requires financial data from data source **408**, dispatcher **402** will first examine an information list. Dispatcher **402** may determine that Page server **404(3)**, for example, has access to the requisite data in data source **408**. Dispatcher **402** will thus route the URL request to Page server **404(3)**. This "connection caching" functionality is described in more detail below, under the heading "Performance." Alternately, Dispatcher **402** also has the ability to determine whether a particular Page server already has the necessary data cached in the Page server's page cache (described in more detail below, under the heading "Performance"). Dispatcher **402** may thus determine that Page server **404(1)** and **404(2)** are both logged into Data source **408**, but that Page server **404(2)** has the financial information already cached in Page server **404(2)**'s page cache. In this case, Dispatcher **402** will route the URL request to Page server **404(2)** to more efficiently process the request.

Finally, Dispatcher **402** may determine that a number or all Page servers **404(1)**–(n) are logged into Data source **408**. In this scenario, Dispatcher **402** can examine the number of requests that each Page server is servicing and route the request to the least busy page server. This "load balancing" capability can significantly increase performance at a busy Web site and is discussed in more detail below, under the heading "Scalability".

If, for example, Page server **404(2)**, receives the request, Page server **404(2)** will process the request. While Page server **404(2)** is processing the request, Web server executable **201**(E) can concurrently process other Web client requests. This partitioned architecture thus allows both Page server **404(2)** and Web server executable **201**(E) to. simultaneously process different requests, thus increasing the efficiency of the Web site. Page server **404(2)** dynamically generates a Web page in response to the Web client request, and the dynamic Web page is then either transmitted back to requesting Web client **200** or stored on a machine that is accessible to Web server **201**, for later retrieval.

One embodiment of the claimed invention also provides a Web page designer with HTML extensions, or "dyna" tags. These dyna tags provide customized HTML functionality to a Web page designer, to allow the designer to build customized HTML templates that specify the source and placement of retrieved data. For example, in one embodiment, a "dynatext" HTML extension tag specifies a data source and a column name to allow the HTML template to identify the data source to log into and the column name from which to retrieve data. Alternatively, "dyna-anchor" tags allow the designer to build hyperlink queries while "dynablock" tags provide the designer with the ability to iterate through blocks of data. Page servers use these HTML templates to create dynamic Web pages. Then, as described above, these dynamic Web pages are either transmitted back to requesting Web client **200** or stored on a machine that is accessible to Web server **201**, for later retrieval.

The presently claimed invention provides numerous advantages over prior art Web servers, including advantages in the areas of performance, security, extensibility and scalability

Performance

One embodiment of the claimed invention utilizes connection caching and page caching to improve performance. Each Page server can be configured to maintain a cache of connections to numerous data sources. For example, as illustrated in FIG. **4**, Page server **404(1)** can retrieve data from data source **406**, data source **408** or data source **410**. Page server **404(1)** can maintain connection cache **412(1)**, containing connections to each of data source **406**, data source **408** and data source **410**, thus eliminating connect times from the Page servers to those data sources.

Additionally, another embodiment of the present invention supports the caching of finished Web pages, to optimize the performance of the data source being utilized. This "page

caching" feature, illustrated in FIG. **4** as Page cache **414**, allows the Web site administrator to optimize the performance of data sources by caching Web pages that are repeatedly accessed. Once the Web page is cached, subsequent requests or "hits" will utilize the cached Web page rather than re-accessing the data source. This can radically improve the performance of the data source.

Security

The present invention allows the Web site administrator to utilize multiple levels of security to manage the Web site. In one embodiment, the Page server can utilize all standard encryption and site security features provided by the Web server. In another embodiment, the Page server can be configured to bypass connection caches **412(1)**–(n), described above, for a particular data source and to require entry of a user-supplied identification and password for the particular data source the user is trying to access.

Additionally, another embodiment of the presently claimed invention requires no real-time access of data sources. The Web page caching ability, described above, enables additional security for those sites that want to publish non-interactive content from internal information systems, but do not want real-time Internet accessibility to those internal information systems. In this instance, the Page server can act as a "replication and staging agent" and create Web pages in batches, rather than in real-time. These "replicated" Web pages are then "staged" for access at a later time, and access to the Web pages in this scenario is possible even if the Page server and dispatcher are not present later.

In yet another embodiment, the Page server can make a single pass through a Web library, and compile a Web site that exists in the traditional form of separately available files. A Web library is a collection of related Web books and Web pages. More specifically, the Web library is a hierarchical organization of Web document templates, together with all the associated data source information. Information about an entire Web site is thus contained in a single physical file, thus simplifying the problem of deploying Web sites across multiple Page servers. The process of deploying the Web site in this embodiment is essentially a simple copy of a single file.

Extensibility

One embodiment of the present invention provides the Web site administrator with Object Linking and Embedding (OLE) 2.0 extensions to extend the page creation process. These OLE 2.0 extensions also allow information submitted over the Web to be processed with user-supplied functionality. Utilizing development tools such as Visual Basic, Visual C++ or PowerBuilder that support the creation of OLE 2.0 automation, the Web site administrator can add features and modify the behavior of the Page servers described above. This extensibility allows one embodiment of the claimed invention to be incorporated with existing technology to develop an infinite number of custom web servers.

For example, OLE 2.0 extensions allow a Web site administrator to encapsulate existing business rules in an OLE 2.0 automation interface, to be accessed over the Web. One example of a business rule is the steps involved in the payoff on an installment or mortgage loan. The payoff may involve, for example, taking into account the current balance, the date and the interest accrued since the last payment. Most organizations already have this type of business rule implemented using various applications, such as Visual Basic for client-server environments, or CICS programs on mainframes. If these applications are OLE 2.0 compliant, the Page server "dynaobject" HTML extension tag can be used to encapsulated the application in an OLE 2.0 automation interface. The Page server is thus extensible, and can incorporate the existing application with the new Page server functionality.

Scalability

One embodiment of the claimed invention allows "plug and play" scalability. As described above, referring to FIG. **4**, Dispatcher **402** maintains information about all the Page servers configured to be serviced by Dispatcher **402**. Any number of Page servers can thus be "plugged" into the configuration illustrated in FIG. **4**, and the Page servers will be instantly activated as the information is dynamically updated in Dispatcher **402**. The Web site administrator can thus manage the overhead of each Page server and modify each Page server's load, as necessary, to improve performance. In this manner, each Page server will cooperate with other Page servers within a multi-server environment. Dispatcher **402** can examine the load on each Page server and route new requests according to each Page server's available resources. This "load-balancing" across multiple Page servers can significantly increase a Web site's performance.

FIG. **5** illustrates the processing of a Web browser request in the form of a flow diagram, according to one embodiment of the presently claimed invention. A Web browser sends a URL request to a Web server in processing block **500**. In processing block **502**, the Web server receives the URL request, and an interceptor then intercepts the handling of the request in processing block **504**. The interceptor connects to a dispatcher and sends the URL request to the dispatcher in processing block **506**. In processing block **508**, the dispatcher determines which Page servers can handle the request. The dispatcher also determines which Page server is processing the fewest requests in processing block **510**, and in processing block **512**, the dispatcher sends the URL request to an appropriate Page server. The Page server receives the request and produces an HTML document in processing block **514**. The Page server then responds to the dispatcher with notification of the name of the cached HTML document in processing block **516**. In processing block **518**, the dispatcher responds to the interceptor with the document name, and the interceptor then replaces the requested URL with the newly generated HTML document in processing block **520**. The Web server then sends the new HTML document to the requesting client in processing block **522**. Finally, the Web browser receives and displays the HTML document created by the Page server at processing block **524**.

Thus, a method and apparatus for creating and managing custom Web sites is disclosed. These specific arrangements and methods described herein are merely illustrative of the principles of the present invention. Numerous modifications in form and detail may be made by those of ordinary skill in the art without departing from the scope of the present invention. Although this invention has been shown in relation to a particular preferred embodiment, it should not be considered so limited. Rather, the present invention is limited only by the scope of the appended claims.

We claim:

1. A computer-implemented method for managing a dynamic Web page generation request to a Web server, said computer-implemented method comprising the steps of:

routing a request from a Web server to a page server, said page server receiving said request and releasing said

Web server to process other requests wherein said routing step further includes the steps of:

intercepting said request at said Web server and routing said request to said page server;

processing said request, said processing being performed by said page server while said Web server concurrently processes said other requests; and

dynamically generating a Web page in response to said request, said Web page including data dynamically retrieved from one or more data sources.

2. The computer-implemented method in claim **1** wherein said step of routing said request includes the steps of:

routing said request from said Web server to a dispatcher; and

dispatching said request to said page server.

3. The computer-implemented method in claim **1** wherein said step of processing said request includes the step of identifying said one or more data sources from which to retrieve said data.

4. The computer-implemented method in claim **1** wherein said step of dynamically generating said Web page includes the step of dynamically retrieving said data from said one or more data sources.

5. The computer-implemented method in claim **1** wherein said step of processing said request includes the step of said page server maintaining a connection cache to said one or more data sources.

6. The computer-implemented method in claim **1** wherein said step of processing said request includes the step of logging into said one or more data sources.

7. The computer-implemented method in claim **1** wherein said step of dynamically generating said Web page includes the step of maintaining a page cache containing said Web page.

8. The computer-implemented method in claim **1** wherein said page server includes tag-based text templates for configuring said Web page.

9. The computer-implemented method in claim **8** wherein said step of processing said request further includes the step of inserting said-dynamically retrieved data from said one or more data sources into said tag-based text templates.

10. The computer-implemented method in claim **8** wherein at least one of said tag-based text templates drives a format of the data dynamically retrieved from said one or more data sources in response to said request.

11. The computer-implemented method in claim **8** wherein said tag-based text templates include HTML templates.

12. The computer-implemented method in claim **1** wherein said step of processing said request further includes the step of dynamically updating data at said one or more data sources.

13. The computer-implemented method in claim **1** wherein said step of processing said request further includes the step of processing an object handling extension.

14. The computer-implemented method in claim **13** wherein said object handling extension is an OLE extension.

15. A computer-implemented method comprising the steps of:

transferring a request from an HTTP-compliant device to a page server, said page server receiving said request and releasing said HTTP-compliant device to process other requests wherein said transferring step further includes the steps of:

intercepting said request at said HTTP-compliant device and transferring said request to said page server;

processing said request, said processing being performed by said page server while said HTTP-compliant device concurrently processes said other requests; and

dynamically generating a page in response to said request, said page including data dynamically retrieved from one or more data sources.

16. The computer-implemented method in claim **15** wherein said step of transferring said request includes the steps of:

transferring said request from said HTTP-compliant device to a dispatcher; and

dispatching said request to said page server.

17. The computer-implemented method in claim **15** wherein said step of processing said request includes the step of identifying said one or more data sources from which to retrieve said data.

18. The computer-implemented method in claim **15** wherein said step of dynamically generating said page includes the step of dynamically retrieving said data from said one or more data sources.

19. The computer-implemented method in claim **15** wherein said step of processing said request includes the step of said page server maintaining a connection cache to said one or more data sources.

20. The computer-implemented method in claim **15** wherein said step of processing said request includes the step of logging into said one or more data sources.

21. The computer-implemented method in claim **15** wherein said step of dynamically generating said page includes the step of maintaining a page cache containing said page.

22. The computer-implemented method in claim **15** wherein said page server includes tag-based text templates for configuring said page.

23. The computer-implemented method in claim **22** wherein said step of processing said request further includes the step of inserting said dynamically retrieved data from said one or more data sources into said tag-based text templates.

24. The computer-implemented method in claim **22** wherein at least one of said tag-based text templates drives a format of the data dynamically retrieved from said one or more data sources in response to said request.

25. The computer-implemented method in claim **22** wherein said tag-based text templates include HTML templates.

26. The computer-implemented method in claim **15** wherein said step of processing said request further includes the step of dynamically updating data at said one or more data sources.

27. The computer-implemented method in claim **15** wherein said step of processing said request further includes the step of processing an object handling extension.

28. The computer-implemented method in claim **27** wherein said object handling extension is an OLE extension.

29. A computer-implemented method comprising the steps of:

transferring a request from an HTTP-compliant device to a dispatcher;

maintaining dynamic information regarding data sources a given page server may access;

dispatching said request to an appropriate page server based on said request and based on said dynamic information, said page server receiving said request and

releasing said HTTP-compliant device to process other requests;

processing said request, said processing being performed by said page server while said HTTP-compliant device concurrently processes said other requests; and

dynamically generating a page in response to said request, said page including data dynamically retrieved from one or more data sources.

* * * * *

A3

[redacted]

Class	Subclass	ISSUE CLASSIFICATION
395	200.33	

5894554

SERIAL NUMBER	FILING DATE	CLASS	SUBCLASS	GROUP ART UNIT	EXAMINER
08/636,477	04/23/96	395	200.33	2782	Kim

APPLICANTS

KEITH LOWERY, RICHARDSON, TX; ANDREW D. LEVINE, PLANO, TX; RONALD L. HOWELL, ROWLETT, TX.

CONTINUING DATA********
VERIFIED

FOREIGN/PCT APPLICATIONS********
VERIFIED

FOREIGN FILING LICENSE GRANTED 06/05/96

Foreign priority claimed / 35 USC 119 conditions met / Verified and Acknowledged	AS FILED	STATE OR COUNTRY	SHEETS DRWGS.	TOTAL CLAIMS	INDEP. CLAIMS	FILING FEE RECEIVED	ATTORNEY'S DOCKET NO.
☐ yes ☑ no / ☐ yes ☑ no / Examiner's initials		TX	5	16	0	$984.00	02577.P001

ADDRESS
JAMES H. SALTER
BLAKELY SOKOLOFF TAYLOR AND ZAFMAN
12400 WILSHIRE BOULEVARD
7TH FLOOR
LOS ANGELES CA 90025

TITLE
~~MANAGING WEB SITES AND DYNAMIC WEB PAGE GENERATION REQUESTS~~
SYSTEM FOR MANAGING DYNAMIC WEB PAGE GENERATION REQUESTS BY INTERCEPTING REQUEST AT WEB SERVER AND ROUTING TO PAGE SERVER THEREBY RELEASING WEB SERVER TO PROCESS OTHER REQUESTS

PTO-436L (Rev.12-94)

PARTS OF APPLICATION FILED SEPARATELY		Applications Examiner
NOTICE OF ALLOWANCE MAILED	Rehana Perveen, Assistant Examiner	CLAIMS ALLOWED: Total Claims 11; Print Claim 1
ISSUE FEE: Amount Due 1210.00; Date Paid 4-19-99	THOMAS G. LEE, SUPERVISORY PATENT EXAMINER, GROUP 2700, Primary Examiner	DRAWING: Sheets Drwg. 5; Figs. Drwg. 5; Print Fig. 4
Label Area	PREPARED FOR ISSUE	ISSUE BATCH NUMBER N34

WARNING: The information disclosed herein may be restricted. Unauthorized disclosure may be prohibited by the United States Code Title 35, Sections 122, 181 and 368. Possession outside the U.S. Patent & Trademark Office is restricted to authorized employees and contractors only.

Form PTO-436A (Rev. 8/92)

ISSUE FEE IN FILE

(FACE)

REMARKS

Applicants respectfully request consideration of the subject application as amended herein. This Amendment is submitted in response to an Office Action dated July 3, 1997. Claims 1-16 stand rejected. In this Amendment, the title of the invention has been replaced and Claims 1 and 16 have been amended

In the July 3, 1997, Office Action, the Examiner objected to the title of the invention as not being descriptive. Applicants have amended the title of the invention to overcome the Examiner's objection. The Examiner also rejected Claims 1-16 under 35 U.S.C. §103 as being unpatentable over Barbari ("Barbari" U.S. Patent No. 5,532,838) in view of Goldberg et al. "Beyond the Web: Excavating the Real World Via Mosaic" ("Goldberg). Applicants respectfully traverse the Examiner's contention that the combination of these references renders the claimed invention unpatentable.

Applicants respectfully contend that neither Barbari nor Goldberg apply to the presently claimed invention. More specifically, Barbari does not teach the invention substantially as claimed, as submitted by the Examiner. Instead, Barbari is directed at a "computerized system for dynamically creating a document from a database responsive to telephonic input from a remote user, and transmitting the document to the user's facsimile machine" (Barbari, Abstract).

Barbari thus simply describes a particular type of database application. The use of databases to store information, and the use of applications to retrieve this information from the database is well known in the art. Additionally, the use of applications to dynamically create lists of information is also well known in the art. Barbari purports to apply this generally well

known techniques to allowing an individual to access a database using a telephone, to direct queries to the database that results not in the transmission of a "canned" or pre-defined message but in a document created and formatted suitable to the individual's specific needs, and then have that document transmitted to a facsimile receiving unit to which the individual has access.

There is no teaching or suggestion in Barbari that remotely resembles the presently claimed invention. Specifically, the present invention is directed at managing Web sites and dynamic Web page generation. The actual creation of dynamic Web pages via CGI applications is known in the art, as described under the heading "Description of related art" in the application (Pages 1-3). Page 3, lines 8-10, describes that "existing development tools are limited in their capabilities to facilitate dynamic Web page generation, and do not address the issue of managing Web requests or Web sites." Thus, the focus of the claimed invention is clearly not in the generation or creation of dynamic Web pages but rather at the management of Web requests or Web sites.

There is no teaching or suggestion in Barbari for managing Web requests or Web sites. Additionally, there is no support in Barbari for the Examiner's contention that Barbari teaches using a "page server" as claimed. Barbari does not teach, for example, receiving a request on a page server and releasing the Web server to process other requests, as claimed. A page server in the context of the presently claimed invention is not simply a server that has "pages" on it, as is the "page server" in Barbari. Instead, as illustrated in Figure 4 and described in the specification on Pages 10-14, a page server according to one embodiment of the presently claimed invention intercepts

Web requests from a Web client and appropriately routes the requests after releasing the Web server to service other requests.

More specifically, the page server interacts with the Web server via an interceptor an a dispatcher. As described in the specification, when a Web request is made to a Web server, the interceptor intercepts the request and routes the request to the dispatcher. The dispatcher receives the request and dispatches the request to one of multiple page servers. The dispatcher maintains a variety of information regarding each page server on the network and dispatches the requests based on this information. There is no teaching or suggestion in Barbari for this type of "page server" or this type of interaction between a Web server and a page server.

The combination of Goldberg with Barbari also does not teach the claimed invention. Goldberg does not remotely address the issue of managing Web requests. Instead, Goldberg simply describes "a teleoperated robot" that allows users to reach beyond the digital boundaries of the Web (Goldberg, Page 1). The Examiner points to a portion of Goldberg that describes a first server communicating with a second server. Communications between servers on a network is known in the art. The claimed invention is not addressed to a first server communicating with a second server but rather to managing requests from a client to a Web server with the use of page servers. The combination of Barbari with Goldberg therefore does not render the claimed invention unpatentable. Applicants respectfully submit that Barbari and Goldberg, alone or in combination, do not apply to the presently claimed invention and therefore do not render the claimed invention unpatentable. As such, and request the Examiner to withdraw the rejection to the claims under 35 U.S.C. §103.

In conclusion, it is respectfully submitted that in view of the amendments and remarks set forth herein, that all objections and rejections have been overcome. All claims are now in condition for allowance and such action is earnestly solicited.

If there are any additional charges, please charge them to our Deposit Account Number 02-2666. If a telephone conference would facilitate the prosecution of this application, the Examiner is invited to contact Jim H. Salter at (408) 720-8598.

Respectfully submitted,
BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP

Date: 10/1, 1997

James H. Salter
Registration Number: 35,668

12400 Wilshire Boulevard
Seventh Floor
Los Angeles, CA 90025-1026
(408) 720-8598

FIRST CLASS CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail with sufficient postage in an envelope addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231 on October 1, 1997
Date of Deposit

Cindy Murphy
Name of Person Mailing Correspondence

Signature

10-1-97
Date

A4

jc518 U.S. PTO 09/234048 01/19/99

Class	Subclass
710	5

ISSUE CLASSIFICATION

PATENT NUMBER
6415335
6415335

U.S. UTILITY PATENT APPLICATION

O.I.P.E.	PATENT DATE
SCANNED AGA Q.A. CS	JAN 02 2002

SECTOR	CLASS	SUBCLASS	ART UNIT	EXAMINER
	710	5	215	Perveen

FILED WITH: ☐ DISK (CRF) ☐ FICHE
(Attached in pocket on right inside flap)

PREPARED AND APPROVED FOR ISSUE

ISSUING CLASSIFICATION

ORIGINAL CLASS	ORIGINAL SUBCLASS	CROSS REFERENCE(S) CLASS	CROSS REFERENCE(S) SUBCLASS (ONE SUBCLASS PER BLOCK)
710	5	710	7
INTERNATIONAL CLASSIFICATION		709	219 223 238
G06F 13/14			
G06F 13/20			
			☐ Continued on Issue Slip Inside File Jacket

☑ TERMINAL DISCLAIMER	DRAWINGS			CLAIMS ALLOWED	
09/234,048	Sheets Drwg.	Figs. Drwg.	Print Fig.	Total Claims	Print Claim for O.G.
	5	5	1	29	1
☐ a) The term of this patent subsequent to _____ (date) has been disclaimed.	Rehana Perveen (Assistant Examiner) 2/7/02 (Date)			NOTICE OF ALLOWANCE MAILED	
☑ b) The term of this patent shall not extend beyond the expiration date of U.S Patent No. 5,894,554.	(Primary Examiner) 2/7/02 (Date)			2-8-02	
				ISSUE FEE	
				Amount Due $1280.00	Date Paid 5-20-02
☐ c) The terminal ___ months of this patent have been disclaimed.	(Legal Instruments Examiner) 2-13-02 (Date)			ISSUE BATCH NUMBER	

WARNING:
The information disclosed herein may be restricted. Unauthorized disclosure may be prohibited by the United States Code Title 35, Sections 122, 181 and 368. Possession outside the U.S. Patent & Trademark Office is restricted to authorized employees and contractors only.

Form PTO-436A
(Rev. 6/98)

(LABEL AREA)

(FACE)

REMARKS

This Application has been carefully reviewed in light of the Office Action mailed February 23, 2001. At the time of the Office Action, Claims 17-45 were pending in this patent application. The Examiner rejected Claims 17-45. Thus, Claims 17-45 are now pending in the Application. Applicants respectfully request reconsideration and favorable action in this case.

IN THE TITLE

The Examiner has rejected the title of the invention as being non-descriptive of the invention to which the claims are directed. Applicants respectfully traverse this rejection, but in the interest of advancing prosecution have amended the title to read "SYSTEM AND METHOD FOR RESPONDING TO REQUESTS ASSOCIATED WITH DYNAMIC WEB PAGE GENERATION". Applicants respectfully submit that the title as amended is descriptive of the invention to which the claims are directed.

IN THE DRAWINGS

Applicants respectfully request that submission of formal drawings be deferred until prior to issuance.

DOUBLE PATENTING

The Examiner has rejected claims 17-45 under the judicially created doctrine of obviousness-type double patenting as being unpatentable over Claims 1-11 of Lowery et al., U. S. Pat. No. 5,894,554, ("Lowery"). Applicants have included a Terminal Disclaimer with respect to the double patenting rejection. Applicants respectfully request withdrawal of this rejection.

SECTION 102 REJECTION

Claims 17-45 stand rejected under U.S.C. § 102(e) as being anticipated by Leaf, U.S. Pat. No. 5,754,772 ("Leaf"). Applicants respectfully traverse this rejection for at least the following reasons.

Claim 17 recites, in part, "intercepting said request at said Web server and routing said request to said page server". Leaf does not teach or suggest "intercepting said request".

Instead, Leaf teaches that the web server routes the request directly to the transaction gateway client. Leaf, col. 4, lines 55-57. Leaf does not teach or suggest "intercepting said request at said Web server" because merely routing a request from a web server to the transaction gateway does not involve interception.

Therefore, for at least this reason, Claim 17 is patentable over Leaf. Thus, Applicants respectfully request allowance of Claim 17.

Dependent Claims 18-30 depend from independent Claim 17, shown above to be allowable. Claims 18-30 are allowable as depending from an allowable base claim and as defining further distinctions over the cited reference.

In particular, Claim 21 recites, in part, "maintaining a connection cache to said one or more data sources" and Claim 23 recites, in part, "maintaining a page cache containing said Web page". The Examiner states that the elements of Claims 21 and 23 are inherent in Leaf. Office Action, p. 4. Leaf does not involve either a connection cache or a page cache. Applicants respectfully submit that neither a connection cache nor a page cache are inherent in Leaf. Applicants respectfully request that the Examiner indicate some teaching or suggestion of Leaf with respect to a page cache and a connection cache. Therefore, Claims 21 and 23 are patentable over Leaf. Thus, Applicants respectfully request allowance of Claims 21 and 23.

Also, Claim 24 recites "wherein said page server includes tag-based text templates for configuring said Web page." Applicants respectfully submit that Leaf does not teach or suggest this element of Claim 24. In contrast to the Examiner's assertion, Leaf merely teaches that the Transaction Gateway Client may format the data into an HTML document. Leaf, col. 4, lines 63-67; See also Leaf, col. 4, lines 25-27. Mere formatting of data into an HTML document does not teach or suggest a "tag-based text template for configuring said Web page". Therefore, Claim 24 is patentable over Leaf. Thus, Applicants respectfully request allowance of Claim 24.

Independent Claim 31 is also allowable at least for the reasons discussed above. In particular, Claim 31 recites "intercepting said request at said HTTP-compliant device and transferring said request to said page server". Accordingly, Applicants respectfully request that Claim 31 be allowed.

Dependent Claims 32-44 depend from independent Claim 31, shown above to be allowable. Claims 32-44 are allowable as depending from an allowable base claim and as defining further distinctions over the cited reference. In particular, Claim 35 recites "wherein said step of processing said request includes the step of said page server maintaining a connection cache to said one or more data sources." Claim 37 recites "wherein said step of dynamically generating said page includes the step of maintaining a page cache containing said page." Claim 38 recites "wherein said page server includes tag-based text templates for configuring said page." Thus, Applicants respectfully request allowance of Claims 32-44.

Independent Claim 45 recites, in part, "maintaining dynamic information regarding data sources a given page server may access". Applicants respectfully submit that Leaf does not teach or suggest this element of Claim 45. Leaf provides no teaching or suggestion of "maintaining dynamic information regarding data sources". Indeed, Leaf notes that "each transaction gateway client pre-establishes a static connection with the n-line transaction processing system." Leaf, col. 2, lines 37-40. Leaf's suggestion of static connections teaches away from "dynamic" information. Therefore, for at least this reason, Claim 45 is patentable over Leaf. Thus, Applicants respectfully request allowance of Claim 45.

REMARKS

This Application has been carefully reviewed in light of the Office Action mailed August 27, 2001. At the time of the Office Action, Claims 17-45 were pending in this patent application. The Examiner objected to Claims 18 and 32 and rejected Claims 17, 19-31, 33-45. Thus, Claims 17-45 are now pending in the Application. Applicants respectfully request reconsideration and favorable action in this case.

Allowable Claims

Applicants thank the Examiner for the indication that Claims 18 and 32 would be allowable if rewritten in independent form including all of the limitations of the base claim and any intervening claims. Applicants respectfully defer rewriting of these Claims until the Examiner has considered the following arguments.

Section 103 Rejection

Claims 17, 19-31, and 33-45 stand rejected under 35 U.S.C. §103(a) as being unpatentable over *Rogers, et al.*, U.S. Pat. No. 5,752,246 ("Rogers"), in view of *Malcolm*, U.S. Pat. No. 5,701,463 ("Malcolm"). Applicants respectfully traverse this rejection for at least the following reasons.

Claim 17 recites, in part, "routing a request from a Web server to a page server". The Examiner admits that Rogers does not teach these elements of Claim 17. Office Action, p. 3. The Examiner relies solely on Malcolm to teach these elements of Claim 17. Office Action, p. 3. Malcolm involves intercepting a file open request at the operating system level and determining whether the identity of the file to be opened should be replaced with the identity of a substitute file to be opened instead. Malcolm, Abstract. Malcolm further notes that whether to permit execution of a file is implemented at the file server by a controlling utility. Malcolm, col. 4, lines 57-59. Malcolm's interception of a file open request and determination of whether to permit execution of a file does not teach or suggest "routing a request from a Web server to a page server" because mere interception and determination do not teach or suggest "routing" the request from one place to another.

Claim 17 further recites, in part, "processing said request, said processing being performed by said page server while said Web server concurrently processes said other requests" and "said page server receiving said request and releasing said Web server to process other requests". Neither Rogers nor Malcolm, either alone or in combination, teach or suggest

these elements of Claim 17. Rogers involves organizing distributed sub-agents as distributed integration solution servers to support a web server. Rogers, Abstract. Rogers also discusses a means for accepting web client requests for information, obtaining data from one or more databases and presenting that information to the web client. Rogers, col. 5, lines 54-61. At no time does Rogers teach or suggest "concurrently" processing other requests or "releasing said Web server to process other requests" because merely retrieving data from multiple sources does not teach or suggest these elements.

In addition, there is no motivation to combine Rogers and Malcolm. Malcolm involves intercepting file open requests at an operating system level to determine whether a substitute file should be opened instead of the original file. Rogers involves a distributed integration solution associated with web servers. Rogers, Abstract. No motivation exists to combine the web server functionality of Rogers with the operating system level functionality of Malcolm. Also, as Rogers is completely unconcerned with dealing with substitute file names, Applicants respectfully submit that the Examiner has made use of impermissible hindsight based on Applicants' disclosure when combining Rogers and Malcolm.

Therefore, for at least these reasons, Claim 17 is patentable over the cited references, either alone or in combination. Thus, Applicants respectfully request allowance of Claim 17.

Independent Claims 31 and 45 are patentable for at least the reasons discussed above in association with independent Claim 17. Thus, Applicants respectfully request allowance of independent Claims 31 and 45.

Dependent Claims 19-30 depend from independent Claim 17, and dependent Claims 33-44 depend from independent Claim 31. Independent Claims 17 and 31 are shown above to be allowable. Therefore, Claims 19-30 and 33-44 are allowable as depending from an allowable base claim and as defining further distinctions over the cited references. Thus, Applicants respectfully request allowance of Claims 19-30 and 33-44.

In particular, dependent Claim 21 recites, in part, "maintaining a connection cache to said one or more data sources" and Claim 23 recites, in part, "maintaining a page cache containing said Web page". The Examiner states that the elements of Claims 21 and 23 are inherent in Rogers. Office Action, p. 4. Rogers does not involve either a connection cache or a page cache. Applicants respectfully submit that neither a connection cache nor a page cache are inherent in Rogers. Applicants respectfully request that the Examiner indicate some teaching or suggestion of Rogers with respect to a page cache and a connection cache. Therefore, Claims

21 and 23 are patentable over Rogers. Thus, Applicants respectfully request allowance of Claims 21 and 23.

Also, Claim 24 recites "wherein said page server includes tag-based text templates for configuring said Web page." Applicants respectfully submit that Rogers does not teach or suggest this element of Claim 24. In contrast to the Examiner's assertion, Rogers merely teaches that data flows between a web server and a decision support system tool and that formatted report results are provided to a web browser. See Rogers, col. 11, lines 64-65 and col. 12, lines 18-24. Mere retrieval and formatting of data does not teach or suggest a "tag-based text template for configuring said Web page". Therefore, Claim 24 is patentable over Rogers. Thus, Applicants respectfully request allowance of Claim 24.

Dependent Claims 35, 37 and 38 are also patentable for at least the reasons discussed above in association with Claims 21, 23 and 24. Thus, Applicants respectfully request allowance of Claims 35, 37 and 38.

A5

US006845505B1

(12) **United States Patent**
Adunuthula et al.

(10) **Patent No.: US 6,845,505 B1**
(45) **Date of Patent: *Jan. 18, 2005**

(54) **WEB REQUEST BROKER CONTROLLING MULTIPLE PROCESSES**

(75) Inventors: **Seshu Adunuthula**, Fremont, CA (US); **Mala Anand**, Palo Alto, CA (US); **Tsung-Jen Chou**, Menlo Park, CA (US); **Shehzaad Nakhoda**, Palo Alto, CA (US); **Raymond Ng**, Foster City, CA (US); **Robert Pang**, Mountain, CA (US); **Ankur Sharma**, Belmont, CA (US); **Matthew Bookman**, Los Gatos, CA (US)

(73) Assignee: **Oracle International Corporation**, Redwood Shores, CA (US)

(*) Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2)

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **08/794,269**

(22) Filed: **Feb. 3, 1997**

(51) **Int. Cl.**[7] **G06F 9/00**

(52) **U.S. Cl.** **718/105**; 718/104

(58) **Field of Search** 395/674, 675, 395/682, 684, 200.56, 200.33; 707/10; 709/104, 105, 302, 304, 226, 203, 102, 201, 206, 219, 225, 229, 238, 239; 718/104, 105, 102

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,918,595	A *	4/1990	Kahn et al.	364/200
5,210,824	A *	5/1993	Putz et al.	395/145
5,212,793	A *	5/1993	Donica et al.	395/700
5,249,290	A *	9/1993	Heizer	395/650

FOREIGN PATENT DOCUMENTS

EP	553560 A2 *	8/1993
EP	00733969	9/1996
EP	0812088 A2	10/1997
WO	WO97/40457	10/1997

OTHER PUBLICATIONS

J. Gray, et al, "Scal Up with TP Monitors", Byte magazine, Apr. 1995, pp. (1–10).*

Merle P et al. "CorbaWeb: A generic object navigator". Computer Networks and ISDN Systems, vol. 28, No. 11, May 1996, p. 1269–1281 XP004018226

Modeling transaction integrity: how CASE tools illustrate the relationships between transactions and data; Frank, Maurice, DBMS, v6, n1, p62(5), Jan. 1993.

(List continued on next page.)

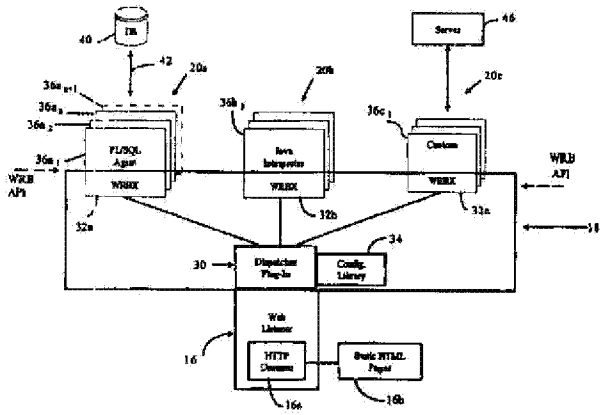
Primary Examiner—Sue Lao
(74) *Attorney, Agent, or Firm*—Christopher J. Brokaw; Hickman Palermo Truong & Becker LLP

(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.

(List continued on next page.)

35 Claims, 6 Drawing Sheets



U.S. PATENT DOCUMENTS

5,329,619	A	*	7/1994	Page et al	395/200
5,341,478	A	*	8/1994	Travis, Jr. et al.	395/200
5,361,350	A		11/1994	Conner et al.	395/600
5,457,797	A		10/1995	Butterworth et al.	395/682
5,504,897	A		4/1996	Gans et al.	395/650
5,546,584	A		8/1996	Lundin et al.	395/700
5,557,798	A	*	9/1996	Skeen et al.	395/650
5,592,654	A	*	1/1997	Djakovic	395/500
5,613,148	A		3/1997	Bezviner et al.	395/800
5,623,656	A		4/1997	Lyons	707/10
5,706,442	A		1/1998	Anderson et al.	395/227
5,708,780	A		1/1998	Levergood et al.	395/200.12
5,715,314	A		2/1998	Payne et al.	380/24
5,724,424	A		3/1998	Gifford	380/24
5,737,592	A		4/1998	Nguyen et al.	395/604
5,737,607	A		4/1998	Hamilton et al.	395/701
5,745,681	A		4/1998	Levine et al.	709/200
5,752,246	A	*	5/1998	Rogers et al.	707/10
5,761,507	A	*	6/1998	Govett	395/684
5,761,662	A		6/1998	Dasan	707/10
5,761,673	A	*	6/1998	Bookman et al.	707/104
5,761,684	A		6/1998	Gibson	707/515
5,774,670	A		6/1998	Montulli	395/200.57
5,778,224	A	*	7/1998	Tobe et al.	395/670
5,796,393	A		8/1998	MacNaughton et al	345/329
5,802,291	A	*	9/1998	Balick et al.	395/200.32
5,805,804	A	*	9/1998	Laursen et al.	395/200.02
5,822,585	A		10/1998	Noble et al	395/680
5,826,239	A		10/1998	Du et al.	705/8
5,826,242	A		10/1998	Montulli	705/27
5,835,712	A		11/1998	DuFresne	709/203
5,848,246	A		12/1998	Gish	395/200.58
5,857,102	A	*	1/1999	McChesney et al.	395/653
5,857,191	A		1/1999	Blackwell et al	707/10
5,859,971	A		1/1999	Bittinger et al.	709/203
5,859,972	A		1/1999	Subramaniam et al.	395/200.33
5,860,072	A		1/1999	Schofield	707/101
5,862,318	A		1/1999	Habben	395/182.18
5,862,325	A		1/1999	Reed et al	
5,864,866	A		1/1999	Henckel et al.	707/103
5,864,871	A		1/1999	Kitain et al.	707/104
5,872,969	A		2/1999	Copeland et al.	395/671
5,875,296	A		2/1999	Shi et al.	395/188.01
5,890,161	A		3/1999	Helland et al.	707/103
5,894,554	A	*	4/1999	Lowery et al.	395/200.33
5,897,622	A		4/1999	Blinn et al	705/26
5,909,492	A		6/1999	Payne et al.	380/24
5,961,601	A		10/1999	Iyengar	709/229
5,991,802	A		11/1999	Allard et al.	709/219
6,067,545	A		5/2000	Wolff	
6,070,191	A	*	5/2000	Narendran et al.	709/26
6,073,241	A		6/2000	Rosenberg et al	
6,098,093	A	*	8/2000	Bayeh et al.	709/203
6,185,625	B1		2/2001	Tso et al.	

OTHER PUBLICATIONS

Luotonen et al., "World–Wide Web proxies", pp. 147–154, computer Network and ISDN system, 01/94.

James Powell, "Creating a hypertext library information system," pp. 59–66, 02/94.

Progress Software, Webspeed 1 0 technical product brief 1998, Wysiwyg://90/http://www.progress–softwa . . ernet/webspeed/white/tech/docs/arch.html.

Computer Reseller News "Progress Software offers tools to speed use of Web", Oct. 7, 1996 http://proquest.umi.com/pdqweb?TS=91798 . . . &Sid=1&Deli= &RQT=309&Dtp=1.

Butler Group Technology Audits, "Butler Group WebSpeed Technology Audit" 1996 http://www.realtime.co.za/web-speed/whitep/wp03.html.

Netscape "Persistent Client State HTTP Cookies" 1997, http://home.netscape com/newsref/std/cookie_spec.html.

M/Gateway Developments Ltd. "Persistence and State Awareness in WebLink" 1996 http://www.intersys.com/products/whitepapers/weblink_state.html.

Oracle Web Application Server™ Installation Guide for Sun SPARC Solaris 2.x , Release 3.0.

Oracle Corporation; Oracle WebServer Architecture; Seshu Adunuthula, Mala Anand, Ankur Sharma; http://www.win.tue.nl/00www/anand.html; dated Apr. 1996; retrieved May 10, 2000.

Distributed Objects on the Internet: Oracle Web Application Server™ 3.0; Richard Delval–Duarte; http://www.fors.com/eoug97/papers/0504.htm; dated Nov. 1996; retrieved May 10, 2000.

Oracle Web Application Server™ Overview, Release 3.0.

Oracle Web Application Server™ Cartridge User's Guide, Release 3.0.

Web Request Broker ™Programmer's Reference, Release 3.0.

Executive Overview; Oracle Web Application Server™ 3.0; http://www.silexsa.com/oracle/was30_eo.htm; retrieved May 11, 2000.

Oracle "Developing Your Own Web Application Server™ Cartridge" Release 3.0.1, published Aug. 14, 1998.

Oracle Web Application Server™, "Installation Guide for Sun SPARC Solaris 2.x" Release 3.0.1, published Aug. 14, 1998.

Oracle Corporation; WRB API Overview; published May 1996 http://www.cs.vu.nl/~eliens/WWW5/papers/Broker.html.

Web Application Server 3.0 "Oracle Web Application Server Documentation Roadmap".

KIVA Software Corporation, "Developing and Managing Web–based Enterprise Applications."

Gray, J., et al.. "Scale Up with TP Monitors," 4465 BYTE, vol. 20, No. 4, Apr. 1995.

Web Application Server 3.0.1 "Overview", published Aug. 14, 1998.

Oracle "Using Oracle Web Application Server™ Cartridge" Release 3.0.1, published Aug. 14, 1998.

Oracle "Performance Tuning", Operating System Parameters (Sun Solaris), published Aug. 14, 1998.

Oracle "Security", "Security Overview", published Aug 14, 1998.

* cited by examiner

ORCL01623091

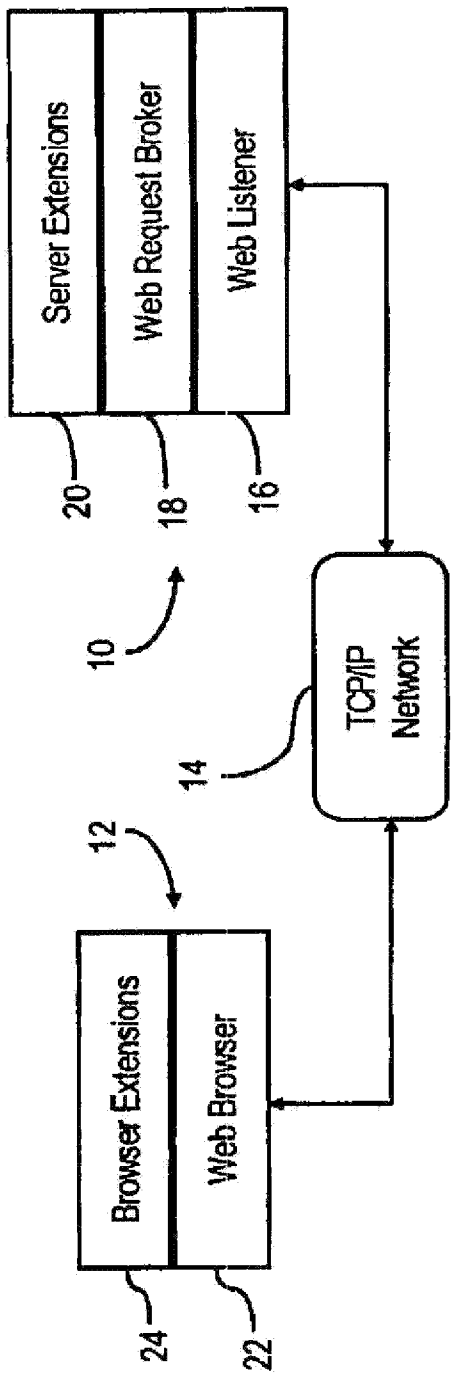

Server Extensions
Web Request Broker
Web Listener
20
18
16
10
14
TCP/IP Network
12
Browser Extensions
Web Browser
24
22

Figure 1

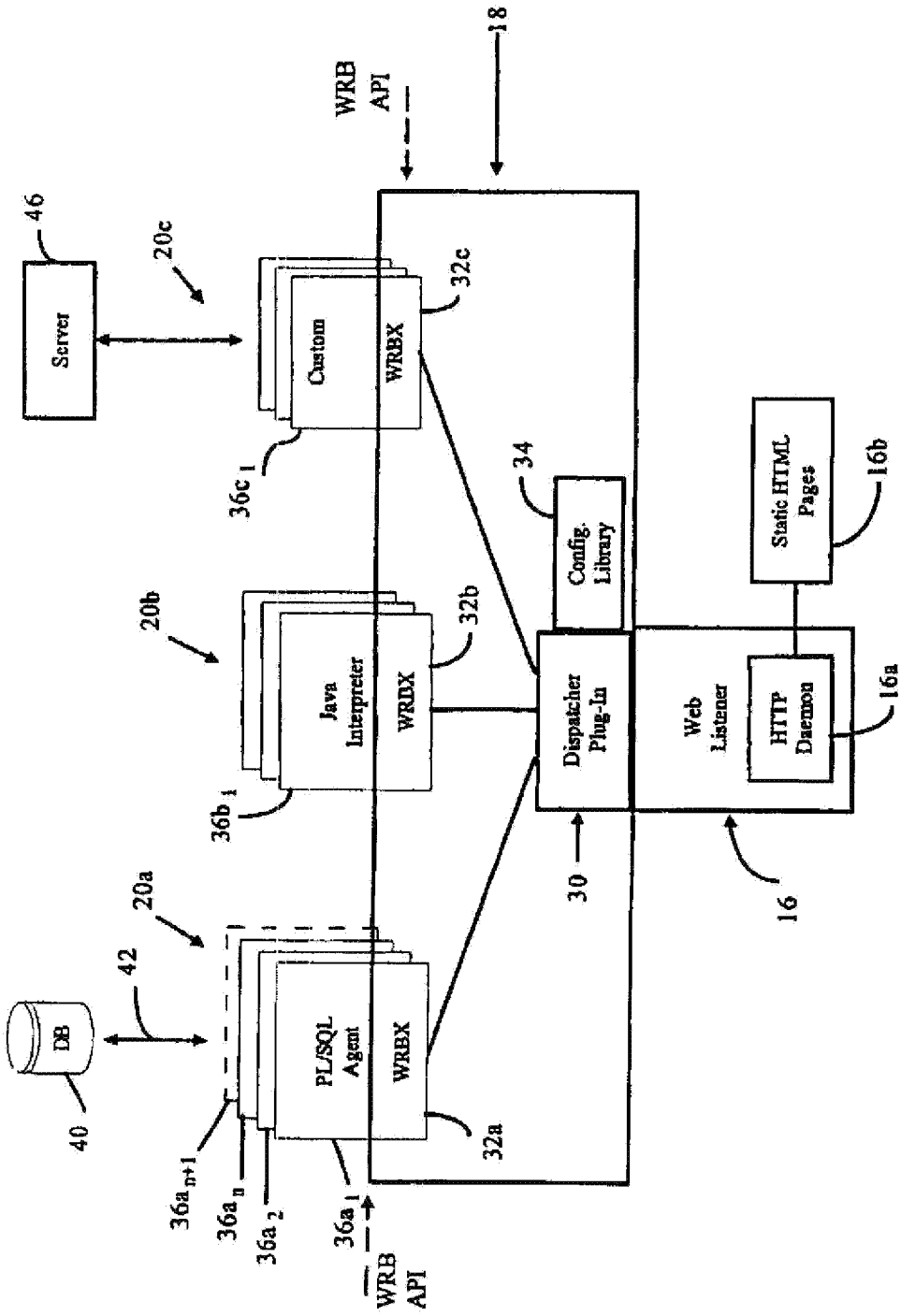

18
WRB API
46
Server
20c
Custom
WRBX
32c
36c 1
34
Config. Library
Static HTML Pages
16b
20b
Java Interpreter
WRBX
32b
36b 1
Dispatcher Plug-In
Web Listener
HTTP Daemon
16a
30
16
20a
42
DB
40
PL/SQL Agent
WRBX
32a
36a n+1
36a n
36a 2
36a 1
WRB API

Figure 2

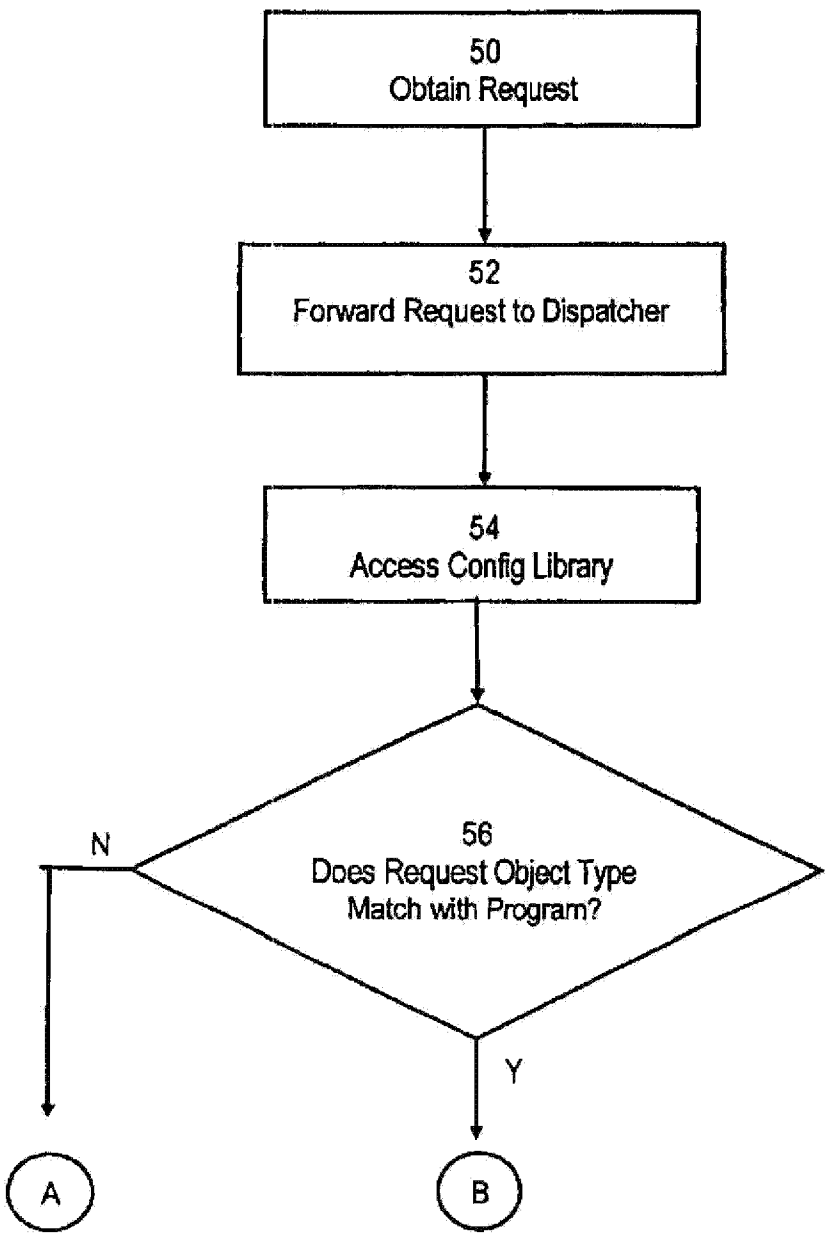

50
Obtain Request
52
Forward Request to Dispatcher
54
Access Config Library
56
Does Request Object Type
Match with Program?
N
Y
A
B

Figure 3A

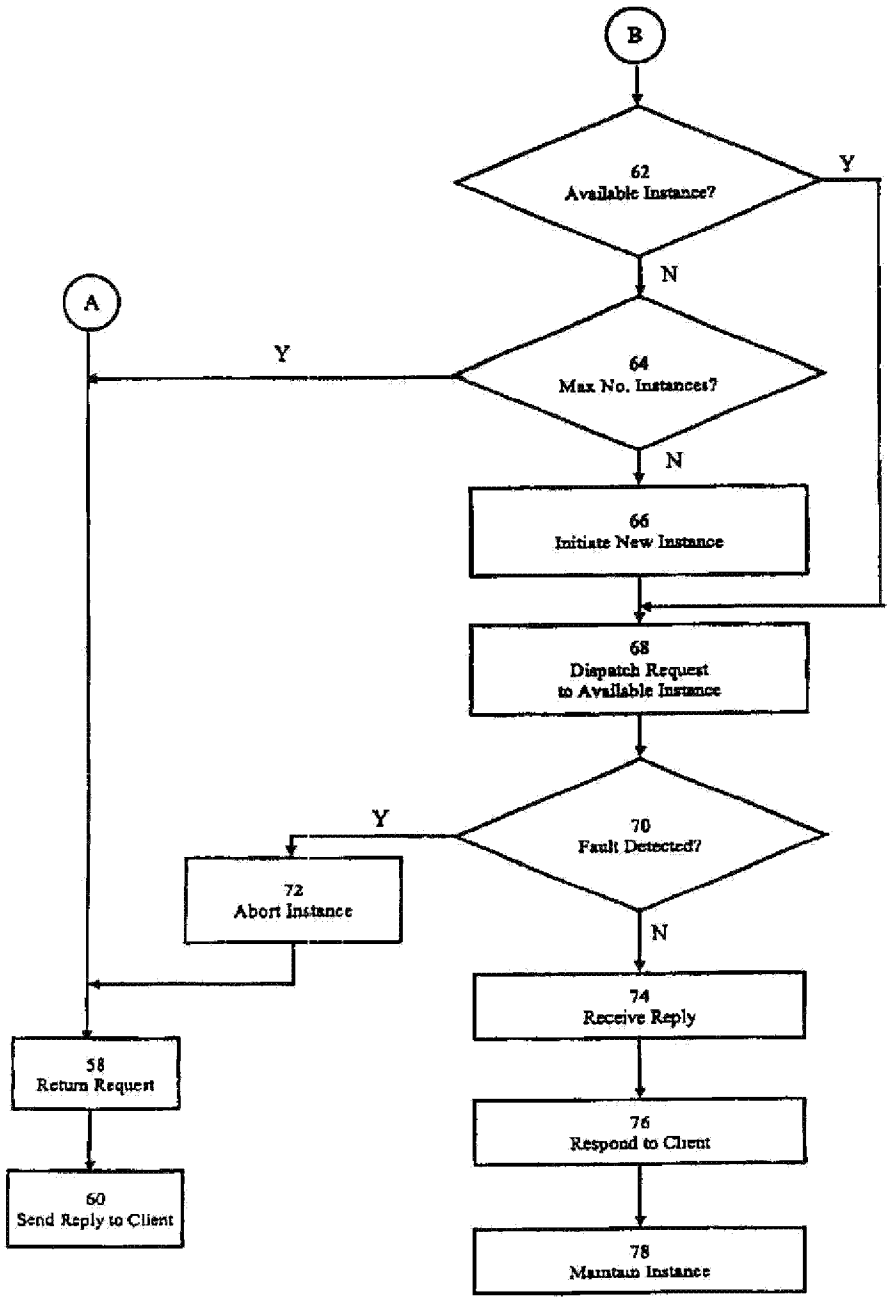

B
62
Available Instance?
Y
N
A
Y
64
Max No. Instances?
N
66
Initiate New Instance
68
Dispatch Request
to Available Instance
Y
70
Fault Detected?
72
Abort Instance
N
74
Receive Reply
76
Respond to Client
78
Maintain Instance
58
Return Request
60
Send Reply to Client

Figure 3B

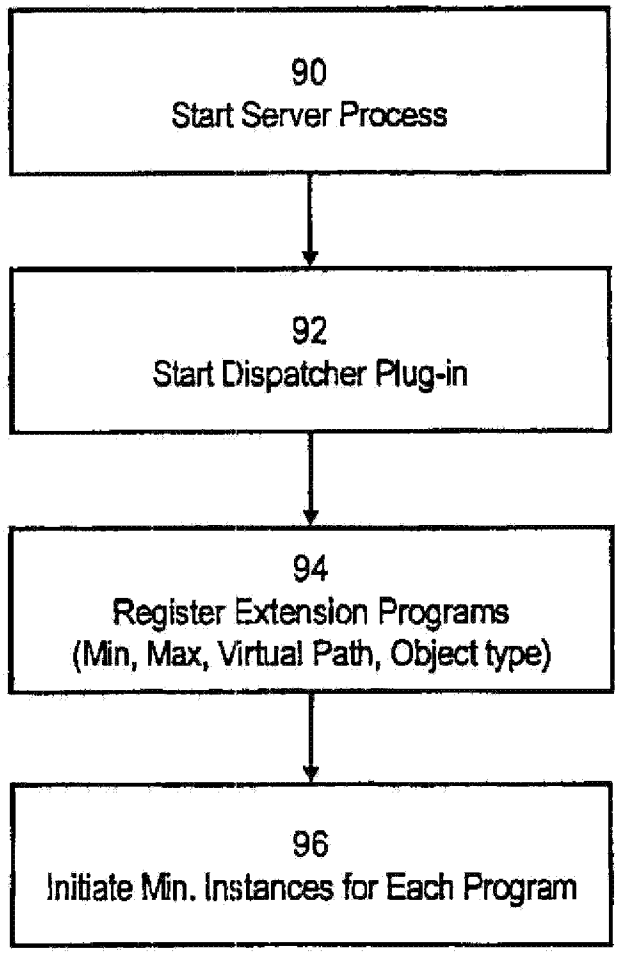

90
Start Server Process
92
Start Dispatcher Plug-in
94
Register Extension Programs
(Min, Max, Virtual Path, Object type)
96
Initiate Min. Instances for Each Program

Figure 4

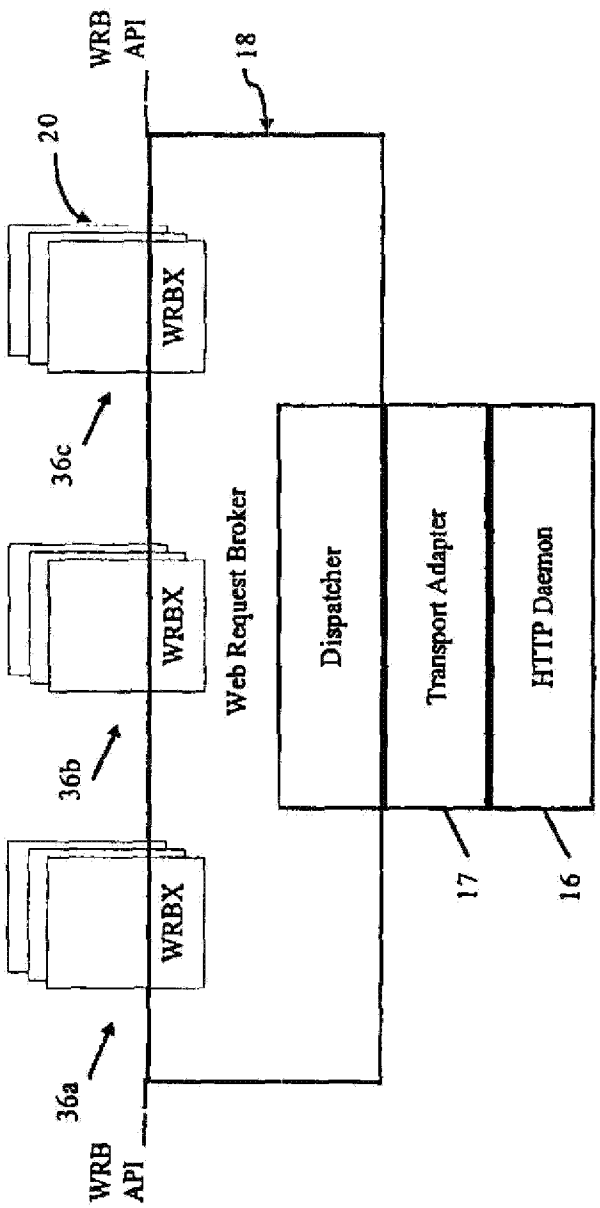

WRB API
18
20
WRBX
36c
WRBX
36b
WRBX
36a
WRB API
Web Request Broker
Dispatcher
Transport Adapter
HTTP Daemon
17
16

Figure 5

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.

ORCL01623098

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:

FIG. **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;

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

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

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

FIG. **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

FIG. **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.

FIG. **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 **16***a* 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 **20***a*, **20***b* and **20***c*. 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 **20***a*, configured to process database queries according to the Oracle-based Programming Language using Structured

ORCL01623099

Query Language (PL/SQL). The PL/SQL agent **20***a* executes a client request having a database query by executing an individual process **36** (i.e., a separate instance of the PL/SQL agent **20***a*). Execution of the instance **36***a* 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 **20***b*, which enable web application developers to write server-side JAVA applications to process client requests. Similarly, a custom server **20***c* 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 **20***a*, **20***b*, and **20***c*, 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 **36***a*, **36***b*, **36***c* for the extension programs **20***a*, **20***b*, **20***c*, 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 **20***a* as the program configured to handle the request. The web request broker **18** will determine whether an existing instance **36***a* of the program **20***a* 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 **36***a* does not exceed a maximum prescribed number.

As shown in FIG. **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 **20***a* is the appropriate extension to process a request, the dispatcher plug-in **30** dispatches the request to the execution engine **32***a*. 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 **32***a* 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 **32***a* then issues a request handler callback function to the specified instance $36a_1$, causing the instance $36a_1$ to process the request, for example by accessing the database **40**. The instance **36***a*, executing the request returns the result to the execution engine **32***a*, 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 **32***a* issues a shutdown function to abort the instance from memory.

Hence, the execution engine **32***a* 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.

FIGS. **3**A and **3**B 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 **36***b* 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 FIG. **3**B). If in step **58** the HTTP daemon **16***a* recognizes the request as a request for a static HTML page, the HTTP daemon accesses the static HTML page from the page memory **16***b*, 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 FIG. **3**B, 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 **20***a*, 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

ORCL01623100

plug-in **30** initiates a new instance of the identified program and dispatches the request to the execution engine **32***a* of the new instance. For example, the dispatcher plug-in **30** initiates a new instance of the PL/SQL agent **20***a*. During this step, the stored sequences of instructions for the PL/SQL agent **20***a* are accessed to create a new instance **36***a*, of the program **20***a* in an address space that is separate from the address space in which dispatcher plug-in **30** is executing.

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

FIG. **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**.

FIG. **5** is a block diagram of the server **10**, according to a second embodiment of the present invention. The first embodiment of FIG. **2** assumes that the dispatcher plug-in **30** is compatible with the lower level processes of the web listener **16** and the HTTP daemon **16***a* The embodiment of FIG. **5** is a modification of the first embodiment of FIG. **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:

said server obtaining the request over a network;

said server identifying, from a plurality of program types available to said server, a program type that corre-

ORCL01623101

sponds to the request, wherein each program type of said plurality of program types implements different functionality than the other program types of said plurality of program types;

determining whether an instance of said program type currently exists that is not currently processing a previously received request;

if no currently executing instance of said program type exists that is not currently processing a previously received request, then said server selectively initiating an instance of the program type based on a comparison between how many instances of said program type are currently executing and a prescribed number of instances of said program type;

said server dispatching the request to said instance of said program type;

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; and

the step of selectively initiating an instance of the program type 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 type.

6. The method of claim **5**, wherein:

the step of selectively initiating an instance of the program type includes the step of initiating a specified minimum number of instances of said program type 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 type 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 type includes the step of returning the request to the network without processing said request when a current number of instances of said program type exceeding a maximum value associated with said program type.

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

11. The method of claim **10**, wherein:

the request identifies a virtual path; and

the step of identifying a program type that corresponds to the request includes the step of identifying the program type 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 exists of a program type configured to handle the request, wherein the available instance is among an existing number of instances of the program type and is not currently processing a previously received request,

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 type for execution of the request if the existing number of instances does not exceed a maximum prescribed number,

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, and

E) registering with the dispatcher a plurality of program types configured to handle respective types of requests, the step of registering with the dispatcher a plurality of said program types including the step of storing in the dispatcher for each of said program types a maximum number of instances and a virtual path specifying an address location associated with the corresponding program type.

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**, wherein each instance is executed within an address space that is separate from the address spaces used by other instances of said program type.

ORCL01623102

18. The method of claim 17, wherein the step of initiating a new instance of the program type 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 type.

19. 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 type configured to handle the request is available among an existing number of instances of the program type, wherein the available instance is not currently processing a previously received request,

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 type for execution of the request if the existing number of instances does not exceed a maximum prescribed number,

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, and

E) delaying deallocation of said new instance at least a predetermined time interval after processing the request for processing a subsequent request, and automatically deallocating said new instance if no subsequent request has arrived after said predetermined time interval.

20. A method, for execution by a server, for responding to a request for performance of an operation by a particular program type, 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 exists of a program type configured to handle the request, wherein the available instance is among an existing number of instances of the program type and is not currently processing a previously received request,

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 type for execution of the request if the existing number of instances does not exceed a maximum prescribed number,

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,

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, said first protocol being one of a plurality of protocols; and

the method further comprises the step of converting the request to a second protocol independent from the first protocol by a transport adapter configured to convert to said second protocol requests received according to any of said first plurality of protocols.

21. The method of claim 20, wherein the existing number of said instances is at least a prescribed minimum number of said instances.

22. The method of claim 20, further comprising the steps of:

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

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

23. The method of claim 20, 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.

24. 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 type, from a plurality of program types, a program type that corresponds to the request, wherein each program type of said plurality of program types implements different functionality than the other program types of said plurality of program types;

determining whether an instance of said program type currently exists that is not currently processing a previously received request;

if no currently executing instance of said program type exists that is not currently processing a previously received request, then selectively initiating an instance of the program type based on a comparison between how many instances of said program type are currently executing and a prescribed number of instances of said program type;

dispatching the request to said instance of said program type;

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

responding to the request based on execution of said instance.

25. The computer readable medium of claim 24, 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.

26. The computer readable medium of claim 24, 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 type.

27. 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

ORCL01623103

network, the request having a prescribed object type specifying an operation to be performed;

a plurality of program types, each program type 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 type, each program type having a prescribed number of instances executing at respective address spaces; and

a dispatcher plug-in configured to identify one of the program types 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 program type based upon the corresponding prescribed number of instances, wherein the available instance of the identified program type is selected to receive said request based on a set of factors that includes that said available instance is not currently processing a previously received request, the dispatcher plug-in sending said response to the network listener based on execution of the request by the available instance.

28. The server of claim **27**, wherein the network listener receives and sends the request and the response based on Hypertext Transfer Protocol (HTTP).

29 The server of claim **28**, 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.

30 The server of claim **27**, 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.

31. The server of claim **27**, further comprising a plurality of execution engines, each execution engine configured to control execution of the instances of the corresponding program types and provide an Application Programming Interface (API) specifying predetermined operations to be performed by the instances of the corresponding program type, 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.

32. The server of claim **31**, 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.

33. The server of claim **27**, wherein each execution engine initiates a prescribed minimum number of said instances of the corresponding program type.

34. The server of claim **27**, wherein the dispatcher plug-in includes a configuration library identifying for each of the program types the corresponding object type and the prescribed number of instances.

35. The server of claim **27**, wherein the object type of the request includes a virtual path specifying the identified one program type.

* * * * *

A6

[redacted]

08/794269 01/31/97

Class	Subclass
718	105

6845505

UTILITY SERIAL NO.	PATENT DATE	PATENT NUMBER
08/794,269	2005	6845505

SERIAL NUMBER	FILING DATE	CLASS	SUBCLASS	GROUP ART UNIT	EXAMINER
		718	105	2126	LA

APPLICANTS

NONE SXP

NONE SXP

CPA

Foreign priority claimed / 35 USC 119 conditions met / Verified and Acknowledged	AS FILED	STATE OR COUNTRY	SHEETS DRWGS.	TOTAL CLAIMS	INDEP. CLAIMS	FILING FEE RECEIVED	ATTORNEY'S DOCKET NO.
☐ yes ☒ no / ☐ yes ☒ no / SXP Examiner's initials	→						

ADDRESS

HICKMAN PALERMO TRUONG & BE
1600 WILLOW STREET
SAN JOSE, CA 95125-5106

TITLE

U.S. DEPT. OF COMM./ PAT. & TM—PTO-436L (Re

PARTS OF APPLICATION FILED SEPARATELY

Applications Examiner

NOTICE OF ALLOWANCE MAILED		CLAIMS ALLOWED	
9/27/04	Assistant Examiner	Total Claims: 35	Print Claim: 17

ISSUE FEE		DRAWING		
Amount Due	Date Paid	Sheets Drwg.	Figs. Drwg.	Print Fig.
	10 18 04	6	6	2

GUE LAO
PRIMARY EXAMINER
Primary Examiner

ISSUE BATCH NUMBER

Label Area

PREPARED FOR ISSUE

WARNING: The information disclosed herein may be restricted. Unauthorized disclosure may be prohibited by the United States Code Title 35, Sections 122, 181 and 368. Possession outside the U.S. Patent & Trademark Office is restricted to authorized employees and contractors only.

Form PTO-436A
(Rev. 6/92)

SEE IN FILE

OC



UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office
Address: COMMISSIONER OF PATENTS AND TRADEMARKS
Washington, D.C. 20231

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.
08/794,269	02/03/97	ANNUNZIATA	3015-046

LMC1/0830

HICKMAN PALERMO TRUONG & BECKER
1600 WILLOW STREET
SAN JOSE CA 95125-5106

EXAMINER	
LAO, L.	
ART UNIT	**PAPER NUMBER**
2755	19

DATE MAILED: 08/30/00

Please find below and/or attached an Office communication concerning this application or proceeding.

Commissioner of Patents and Trademarks

PTO-90C (Rev. 2/95) U.S. G.P.O. 1999 460-693 1- File Copy

ORCL01623263

Office Action Summary	Application No. 08/794,269	Applicant(s) Adunuthula, et al
	Examiner S. Lao	Group Art Unit 2755

[X] Responsive to communication(s) filed on *Jun 8, 2000*

[ ] This action is **FINAL.**

[ ] Since this application is in condition for allowance except for formal matters, **prosecution as to the merits is closed** in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11; 453 O.G. 213.

A shortened statutory period for response to this action is set to expire *3* month(s), or thirty days, whichever is longer, from the mailing date of this communication. Failure to respond within the period for response will cause the application to become abandoned. (35 U.S.C. § 133). Extensions of time may be obtained under the provisions of 37 CFR 1.136(a).

Disposition of Claim

[X] Claim(s) *1-12 and 14-36* is/are pending in the applicat

Of the above, claim(s) _____ is/are withdrawn from consideration

[ ] Claim(s) _____ is/are allowed.

[X] Claim(s) *1-12 and 14-36* is/are rejected.

[ ] Claim(s) _____ is/are objected to.

[ ] Claims _____ are subject to restriction or election requirement.

Application Papers

[ ] See the attached Notice of Draftsperson's Patent Drawing Review, PTO-948.

[ ] The drawing(s) filed on _____ is/are objected to by the Examiner.

[ ] The proposed drawing correction, filed on _____ is [ ] approved [ ] disapproved.

[ ] The specification is objected to by the Examiner.

[ ] The oath or declaration is objected to by the Examiner.

Priority under 35 U.S.C. § 119

[ ] Acknowledgement is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d).

[ ] All [ ] Some* [X] None of the CERTIFIED copies of the priority documents have been

[ ] received.

[ ] received in Application No. (Series Code/Serial Number) _____.

[ ] received in this national stage application from the International Bureau (PCT Rule 17.2(a)).

*Certified copies not received. _____

[ ] Acknowledgement is made of a claim for domestic priority under 35 U.S.C. § 119(e).

Attachment(s)

[X] Notice of References Cited, PTO-892

[ ] Information Disclosure Statement(s), PTO-1449, Paper No(s). _____

[ ] Interview Summary, PTO-413

[ ] Notice of Draftsperson's Patent Drawing Review, PTO-948

[ ] Notice of Informal Patent Application, PTO-152

--- SEE OFFICE ACTION ON THE FOLLOWING PAGES ---

DETAILED ACTION

1. Claims 1-12, 14-36 are pending. This action is in response to applicant's response filed 6/8/2000.

2. The text of those sections of Title 35, U.S. Code not included in this action can be found in a prior Office action.

3. Claims 1, 25 are rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al in view of Heizer and Kahn et al.

As to claim 1, Putz teaches (abstract, col. 9, lines 21-30; col. 10, lines 20-37; col. 12, lines 33-55) a method for responding to a request (remote procedure calls) issued to a server (main server program 55) from a client (client 52-54) over a network system, the method comprising the steps of:

the server obtaining the request over a network (receive remote procedure calls);

the server identifying (check program dictionary) a program type corresponding to the request (named database operation program) among a plurality of program types (operation programs NewDocDecs, Render, DescriptionSearch),

if no currently executing instance of the program type (named operation program) exists, selectively initiating an instance of the program type (fork a child process of the named operation program),

the server dispatching the request to the instance (fork, then pass the remaining operating program arguments),

executing the instance to process the request (invoke operating program for performing),

responding to the request (return results of the operation).

Putz does not teach (1) the step of initiating is based on a comparison between how many instances are executing of the program type is executing and a prescribed number

ORCL01623265

of the program type, (2) determining whether an instance of the program type currently exists that is not currently processing a previously received request.

As to (1), Heizer teaches a method for responding to a request (request) issued to a server program type (server 122) from a client (client 103), including initiating an instance of the program type (server process 202, 203) based on a comparison between how many instances are executing of the program type is executing (fig. 3) and a prescribed number of instances of the program (low/high water mark); see col. 2, line 20 - col. 4, line 55; fig.s 1-5.

Putz creates an instance of a named program type for each client request of that named program type, which eventually will cause server overloading. This the aspect of the request dispatching that Heizer addresses to improve. (Col. 1, lines 21-41). Therefore, one of the ordinary skill in the art would have been motivated to apply the teaching of Heizer to Putz's dispatching of each named program, to prevent server overloading. The combination would have provided maximum number of instances for each named program type.

As to (2), Kahn teaches management of server resource (internal reader), wherein if there is a server instance currently exists (previously created internal reader) that is not currently processing a previously received request (which has finished processing and waiting for additional work, idle), an incoming request/job is assigned to such server instance. Otherwise, new server instances are initiated/created to handle the request/job. See abstract, col. 13, lines 40-63, col. 15, lines 35-61, col. 18, lines 11-35. Kahn determines whether there is an available/idle/not_currently_processing server instance by checking the IDLE flag for an internal reader. It would have been obvious to include the teaching of Kahn into Putz as modified so as to provide handling of changing work flow without excessive system overhead (Kahn, col. 18, lines 31-35).

As to claim 25, it is program product claim of claim 1.

4. Claims 2-3 are rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al, Heizer and Kahn et al and further in view of Donica et al.

ORCL01623266

As to claim 2, Donica teaches responding to a client request, including obtaining the request by a server process (scheduler) executing in a first address space (scheduler address space 30), initiating an instance of a service program (initiator) in a second address space that is separate from the first address space (initiator address space 40 separate from 30), see col. 5, lines 31-34. Since Putz, Heizer, Kahn et al and Donica address load balancing and reducing operating system overhead, it would have been obvious to combine the teachings.

As to claim 3, Donica teaches after executing an instance, maintaining it in memory for executing a subsequent request (idle initiators already assigned to a class), see col. 7, lines 19-24. Note discussion of claim 2 for motivation to combine.

5. Claims 4, 26 are rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al, Heizer and Kahn et al as applied to claims 1, 25 and further in view of Tobe et al.

As to claim 4, Tobe teaches network resource management, including detecting (detect by the system) a fault in an instance during execution thereof (node goes down); and aborting the instance in response to detecting the fault (back up node takes over), see col. 9, lines 42-52. Since Putz, Heizer, Kahn et al and Tobe address resource management and overhead, it would have been obvious to combine the teachings.

As to claim 26, it is program product claim of claim 4.

6. Claims 5-9, 27 are rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al, Heizer and Kahn et al as applied to claims 1, 25 and further in view of Donica et al and Atkins et al.

As to claim 5, Donica teaches specifying at least one of a maximum (maximum number of initiators) and a minimum (minimum number of initiators) number of instances (col. 6, lines 3-5, fig. 2).

As to that each instance executes within an address space that is separate from the address spaces used by other instances of the program, it is taught by Atkins in that each node holds an instance (copy) of a service program (shared object), see pg. 3, left col.,

ORCL01623267

lines 7-20. Since Putz, Heizer, Donica and Atkins address server resource management, it would have been obvious to combine the teachings.

As to claims 6-9, Heizer teaches initiating a specified minimum number of instances (one server process 202) prior to obtaining the request over the network (on start up, col. 3, lines 16-17), after obtaining the request over the network, determining if one of the minimum number of instances is available for processing said request (first request, col. 4, lines 3-13), 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 (third request, col. 4, lines 17-29), executing said instance includes executing an available instance from the minimum number of instances (first request, col. 4, lines 3-13), returning the request to the network without processing said request based on the prescribed number of said instances exceeding a maximum value (21st request, col. 4, lines 52-56).

As to claim 27, it is program product claim of claim 5.

7. Claims 10-11 are rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al, Heizer and Kahn et al as applied to claim 1 and further in view of Donica et al and Travis et al.

As to claims 10-11, Donica teaches specifying a maximum number of instances, as discussed on claim 5. Travis teaches registering a plurality of programs (applications) with the server (register, col. 17, line 58 - col 18, line 45; server registration, col. 18), specifying a virtual path for a program (names of classes/messages), request (invocation, message) identifies a virtual path (message names), identifying the program (method to perform the operation) based on the virtual path identified in the request (select from the database, Invoker Operation, col.s 25-29), Since Putz, Heizer, Kahn et al, Donica and Travis address request allocation, it would have been obvious to combine the teachings.

Travis teaches specifying other data such as performance information (parameters supported, col. 18, lines 20-29), it would have been obvious to specify a maximum number of instances which is commonly regarded as performance information.

ORCL01623268

8. Claims 12, 21-24 are rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al, Heizer and Kahn et al as applied to claim 1 and further in view of Djakovic.

As to claim 12, Heizer teaches receiving the request from a transport protocol process (protocol/network software 113), but fails to teach receiving is according to a first protocol and converting the request to a second protocol independent from the first protocol.

However, Djakovic teaches receiving a request (job submitted by client) according to a first protocol (source protocol) and converting the request to a second protocol independent from the first protocol (form a new job in target protocol), see col. 1, lines 40-51; fig. 3. Since Putz, Heizer, Kahn et al and Djakovic address server processing clients request/job, it would have been obvious to combine the teachings.

As to claim 21, note discussion of claim 1 for obtaining, step A)-C), and note discussion of claim 12 for a transport adaptor / means configured to perform protocol conversions. Heizer further teaches forwarding the request to a dispatcher (control process 204 accepts new client) executed by the server; and processing the request by causing the dispatcher to perform the step 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 (when the maximum number of server processes is set at 20, request 21st is rejected, step 317), see col. 3, lines 10-23; col. 4, lines 3-56. Note discussions of claims 1, 12 for motivations to combine.

As to claim 22, note discussion of claim 6.

As to claim 23, Heizer teaches sending a replying indicating the request is not processed (reject, step 317), and sending such a reply when no program configured to handle the request would have been an obvious condition to apply.

As to claim 24, Heizer teaches server process (control process 204), and plug-in routine added to the server process (library 121) .

ORCL01623269

9. Claims 28-30, 34, 36 are rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al, Heizer, Kahn et al as applied to claim 1 and further in view of admitted prior art (pages 1-3).

As to claim 28, note discussion of claim 1, in particular, note claim 1 for receiving request and sending response, for dispatcher selectively dispatching / dispatching and sending the response / responding, for not currently processing and each program having a prescribed number of instances / maximum / minimum. The instances (server process) of a program type of Heizer reside at respective address spaces (202, 203) since they execute on their respective clients.

Further, Heizer teaches network listener (protocol/network software 113), each program type configured to perform an operation that generates an output in response to receiving a request (a server process is to receive a client request, process the request and send a response).

As to that the dispatcher is in a plug-in format, it is taught by admitted prior art (plug-in extensions, page 2, last para.). It would have been obvious to include the plug-in format of admitted prior art into the system/method of Heizer as modified to provide dynamic response to request (page 2, last para.).

As to claims 29-30, admitted prior art teaches receiving the request and sending the response based on Hypertext Transfer Protocol (HTTP) (HTTP, page 1, lines 9-10), to output a static Hypertext Markup Language (HTML) page (page 1, last para.). A daemon is a well known means for performing services.

As to claim 34, Heizer teaches initiating a prescribed minimum number of said instances of the corresponding program (one instance, see rejection of claim 13). Each execution engine is taught by well known distributed network managers.

As to claim 36, Putz teaches registering (the program directory) program types (named database operations) with a dispatcher (main server program 55) for dispatching client requests. (col. 10, lines 31-37). Typically registering with a program directory includes registering the locations/addresses.

ORCL01623270

10. Claim 35 is rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al, Heizer, Kahn et al in view of admitted prior art as applied to claim 28 and further in view of Donica.

As to claim 35, Donica teaches configuration library (PARMLIB) identifying for each of the programs the corresponding object type (class) and the prescribed number of instances (maximum and minimum numbers of initiators), see col. 6, lines 1-5; col. 8, lines 27-31. As to plug-in format, note discussion of claim 28. Note rejection of claim 2 for motivation to combine.

11. Claim 31 is rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al, Heizer, Kahn et al in view of admitted prior art as applied to claim 28 and further in view of Djakovic.

As to claim 31, note the discussion of claim 12.

12. Claim 32-33 are rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al, Heizer, Kahn et al in view of admitted prior art as applied to claim 28 and further in view of Page et al.

As to claims 32-33, Page teaches a plurality of execution engines to control the execution of program instances and to receive the request (adapters to DCE, DDE, LU6.2) (distributed brokers), API specifying predetermined operations, including at least one of initialization (REGISTER), execution of the request (SEND), and shutdown (EOC, ABENDS), see fig.s 8, 17B, 21, 23. Since Putz, Heizer, Donica, Atkins, APA, Travis and Page address client server management, it would have been obvious to combine the teachings.

13. Claims 14, 17 are rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al, Heizer, Kahn et al as applied to claim 1 and further in view of Donica et al.

As to claim 17, note discussions of claims 1, 10-11 (with respect to Donica), and Heizer further teaches forwarding the request to a dispatcher (control process 204 accepts

ORCL01623271

new client) 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 (decides which of 202, 203 should handle the client), B) if an available instance is available, then dispatching the request for execution by the available instance (first request is given to 202 since it is activated, step 307), 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 (for the third request, a new server process 203 is started, step 313), 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 (when the maximum number of server processes is set at 20, request 21st is rejected, step 317), see col. 3, lines 10-23; col. 4, lines 3-56.

Further, Putz teaches registering (the program directory) program types (named database operations) with a dispatcher (main server program 55) for dispatching client requests. (col. 10, lines 31-37). Typically registering with a program directory includes registering the locations/addresses. Registering with a dispatcher (control process) information of a maximum number of instances (high water mark) of a service is taught by Heizer (see discussion of claim 1).

As to claim 14, Heizer teaches receiving by the dispatcher (control process/server program) a reply (response) from the instance executing and sending information contained to a client (send response). See fig. 1.

14. Claims 15-16 are rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al, Heizer, Kahn et al in view of Donica et al as applied to claim 17 and further in view of Tobe et al.

As to claim 15, note discussion of claim 4 for detecting and terminating/aborting and dispatcher (management node, col. 4, lines 49-51).

As to claim 16, sending a reply is covered by claim 17, step D).

ORCL01623272

15. Claims 18-19 are rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al, Heizer, Kahn et al in view of Donica et al as applied to claim 17 and further in view of Atkins et al.

As to claims 18-19, note discussion of claim 5.

16. Claim 20 is rejected under 35 U.S.C. 103(a) as being unpatentable over Putz, Heizer, Kahn in view of McChesney et al.

As to claim 20, note discussions of claims 1, and Heizer further teaches forwarding the request to a dispatcher (control process 204 accepts new client) 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 (decides which of 202, 203 should handle the client), B) if an available instance is available, then dispatching the request for execution by the available instance (first request is given to 202 since it is activated, step 307), 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 (for the third request, a new server process 203 is started, step 313), 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 (when the maximum number of server processes is set at 20, request 21st is rejected, step 317), see col. 3, lines 10-23; col. 4, lines 3-56.

As to E), McChesney teaches a process/object life cycle management system, including delaying and automatically deallocating an instance after a predetermined time interval (API Shutdown 223, automatically shutdown an idle process after a time interval). See fig. 2, col. 30, line 52 - col. 31, line 3. Since McChesney addresses object/resource management as does Putz as modified, it would have been obvious to combine the teachings.

ORCL01623273

17. Applicant's arguments filed 6/8/2000 have been considered but are moot in view of the new ground(s) of rejection.

In the response, applicant's representative argued in substance that the motivations to combine the cited references must be expressly found in the cited references and not based on hindsight/applicant's disclosure.

The examiner's response is as follows.

MPEP § 2143.01 requires that the suggestion or motivation to combine references must be found in the references themselves, OR, in the knowledge generally available to one of ordinary skill in the art. The proper test for the relevance of a cited combination of references is: "whether the teachings of the prior art, taken as a whole, would have made obvious the claimed invention," In re Gorman, 933 F.2d at 986, 18 USPQ2d at 1888. Subject matter is unpatentable under section 103 if " 'would have been obvious ... to a person having ordinary skill in the art.' While there must be some teaching, reason, suggestion, or motivation to combine existing elements to produce the claimed device, it is not necessary that the cited references or prior art specifically suggest making the combination: In re Nilssen, 851 F.2d 1401, 1403, 7 USPQ2d 1500, 1502 (Fed. Cir. 1988)." Such suggestion or motivation to combine prior art teachings can derive solely from the existence of a teaching, which one of ordinary skill in the art would be presumed to know, and the use of that teaching to solve the same [or] similar problem which it addresses. In re Wood, 599 F.2d 1032, 1037, 202 USPQ 171, 174 (CCPA 1979). "In stun, it is off the mark for litigants to argue, as many do, that an invention cannot be held to have been obvious unless a suggestion to combine prior art teachings is found in a specific reference." In re Oetiker, 24 USPQ2d 1443 (CAFC 1992).

In response to Applicant's argument that the Examiner's conclusion of obviousness is based upon improper hindsight reasoning, it must be recognized that any judgement on obviousness is in a sense necessarily a reconstruction based upon hindsight reasoning. But so long as it takes into account only knowledge which was within the level of ordinary skill at the time the claimed invention was made, and does not include knowledge gleaned

only from the applicant's disclosure, such a reconstruction is proper. In re McLaughlin, 443 F.2d 1392; 170 USPQ 209 (CCPA 1971).

18. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Sue Lao whose telephone number is (703) 305-9657. The fax number for this Group is (703) 305-9731.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the Group receptionist whose telephone number is (703) 305-3900.

Sue Lao
August 23, 2000

MAJID BANANKHAH
PRIMARY EXAMINER

ORCL01623275

Notice of References Cited

Application No.	Applicant(s)		
08/794,269	Adunuthula, et al		
Examiner		Group Art Unit	Page 1 of 1
S. Lao		2755	

U.S. PATENT DOCUMENTS

	DOCUMENT NO.	DATE	NAME	CLASS	SUBCLASS
A	5,894,554	4/1999	Lowery et al	395	200.33
B	6,070,191	5/2000	Narendran et al	709	226
C	6,098,093	8/2000	Bayeh et al	709	203
D					
E					
F					
G					
H					
I					
J					
K					
L					
M					

FOREIGN PATENT DOCUMENTS

	DOCUMENT NO.	DATE	COUNTRY	NAME	CLASS	SUBCLASS
N						
O						
P						
Q						
R						
S						
T						

NON-PATENT DOCUMENTS

	DOCUMENT (Including Author, Title, Source, and Pertinent Pages)	DATE
U		
V		
W		
X		

A7

US006334114B1

(12) **United States Patent**
Jacobs et al.

(10) **Patent No.: US 6,334,114 B1**
(45) **Date of Patent: *Dec. 25, 2001**

(54) **METHOD AND APPARATUS FOR PERFORMING TRANSACTIONS IN A STATELESS WEB ENVIRONMENT WHICH SUPPORTS A DECLARATIVE PARADIGM**

(75) Inventors: **Lawrence Jacobs**, Redwood Shores; **Seshu Adunuthula**, Foster City; **Mala Anand**, Palo Alto, all of CA (US)

(73) Assignee: **Oracle Corporation**, Redwood Shores, CA (US)

(*) Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **08/962,536**

(22) Filed: **Oct. 31, 1997**

(51) **Int. Cl.**[7] **G06F 17/60**

(52) **U.S. Cl.** **705/26**; 705/27; 707/10; 707/104; 395/674; 395/675; 395/682; 395/684; 395/200.56; 395/200.33

(58) **Field of Search** 705/26, 27; 395/200.12, 395/200.57

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,918,595	4/1990	Kahn et al.	364/200
5,210,824	5/1993	Putz et al.	395/145
5,212,793	5/1993	Donica et al.	395/700
5,249,290	9/1993	Heizer	395/650
5,329,619	7/1994	Page et al.	395/200
5,341,478	8/1994	Travis, Jr. et al.	395/200
5,361,350	11/1994	Conner et al.	395/600

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

553560A2	8/1993	(EP) .
0 733 969 A1	9/1996	(EP) .
0812088A2	12/1997	(EP) .

OTHER PUBLICATIONS

Progress Software, "Webspeed 1.0 technical product brief" 1998, wysiwyg://90/http://www.progress-softwa . . . ernet/webspeed/white/tech/docs/arch.html.*

Computer Reseller News "Progress Software offers tools to speed use of Web", Oct. 7, 1996, http://proquest.umi.com/pqdweb?TS=91798 . . . &Sid=1&Deli=1&RQT=309&Dtp=1.*

Butler Group Technology Audits, "Butler Group WebSpeed Technology Audit" Nov. 1996, http://www.realtime.co.za/webspeed/whitep/wp03.html.*

Netscape "Persistent Client State HTTP Cookies" 1997, http://home.netscape.com/newsref/std/cookie_spec.html.*

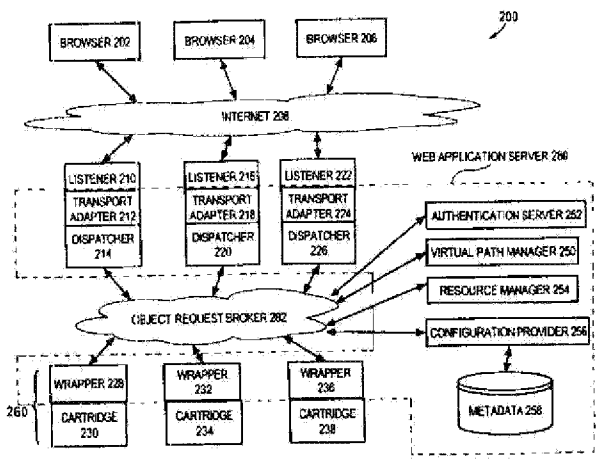
(List continued on next page.)

Primary Examiner—James P. Trammell
Assistant Examiner—Yehdega Retta
(74) *Attorney, Agent, or Firm*—Hickman Palermo Truong & Becker LLP; Carl L. Brandt

(57) **ABSTRACT**

A method and system for processing multiple-request transactions in a stateless environment is provided. A cartridge execution engine intercepts browser messages directed to a cartridge. The cartridge execution engine determines whether the browser messages are associated with transactions. If it is determined that browser messages are associated with transactions, then the cartridge execution engine sends transaction control messages to a transaction manager. In addition, the cartridge execution engine sends operation messages to the cartridge. The cartridge then performs the operations specified in the operation messages. In response to the transaction control messages from the cartridge execution engine, the transaction manager causes the multiple-request transactions to be either committed or rolled back as an atomic unit of work.

48 Claims, 16 Drawing Sheets



U.S. PATENT DOCUMENTS

5,457,797		10/1995	Butterworth et al.	395/682
5,504,897		4/1996	Gans et al.	395/650
5,546,584		8/1996	Lundin et al.	395/700
5,592,654		1/1997	Djakovic	395/500
5,613,148		3/1997	Bezviner et al.	395/800
5,623,656		4/1997	Lyons	707/10
5,706,442		1/1998	Anderson et al.	395/227
5,708,780		1/1998	Levergood et al.	
5,715,314		2/1998	Payne et al	
5,724,424		3/1998	Gifford	
5,737,592		4/1998	Nguyen et al.	395/604
5,737,607		4/1998	Hamilton et al.	395/701
5,745,681		4/1998	Levine et al.	709/200
5,752,246		5/1998	Rogers et al.	707/10
5,761,507		6/1998	Govett	395/684
5,761,662		6/1998	Dasan	707/10
5,761,673		6/1998	Bookman et al.	707/104
5,761,684		6/1998	Gibson	707/515
5,774,670	*	6/1998	Montulli	395/200.57
5,778,224		7/1998	Tobe et al.	395/670
5,796,393		8/1998	MacNaughton et al.	345/329
5,802,291		9/1998	Balick et al.	395/200.32
5,822,585		10/1998	Noble et al.	395/680
5,826,239		10/1998	Du et al.	705/8
5,826,242	*	10/1998	Montulli	705/27
5,835,712		11/1998	DuFresne	709/203
5,857,102		1/1999	McChesney et al.	395/145
5,857,191		1/1999	Blackwell et al.	707/10
5,859,971		1/1999	Bittinger et al.	709/203
5,860,072		1/1999	Schofield	707/101
5,862,318	*	1/1999	Habben	395/182.18
5,862,325		1/1999	Reed et al.	395/200.31
5,864,866		1/1999	Henckel et al.	707/103
5,864,871		1/1999	Kitain et al.	707/104
5,872,969		2/1999	Copeland et al.	
5,875,296		2/1999	Shi et al.	
5,890,161	*	3/1999	Helland et al.	707/103
5,894,554		4/1999	Lowery et al.	395/200.33
5,961,601		10/1999	Iyengar	
5,991,802		11/1999	Allard et al.	
6,070,191		5/2000	Narendran et al.	709/226

OTHER PUBLICATIONS

M/Gateway Developments Ltd. "Persistence and State Awareness in WebLink" 1996, http://www.intersys.com/products/whitepapers/weblink_state.html.*

Oracle Corporation; Oracle WebServer Architecture; Seshu Adunuthula, Mala Anand, Ankur Sharma; http://www.win-.tue.nl/00www/anand.html, Apr. 1996.*

Distributed Objects on the Internet: Oracle Web Application Server [tm] 3.0; Richard Delval--Duarte; Nov. 1996.*

Executive Overview; Oracle Web Application Server TM 3.0.*

Oracle Web Application Server TM Cartridge user's Guide; Release 3.0, 1996/1997.*

Web Request Broler TM Programmer's Reference Release 3 0, 1996/1997.*

Oracle Web Application Server TM Overview Release 3.0, 1996/1997.*

Using Oracle Web Application Server TM Cartridge Release 3.0.1, Apr. 1996.*

Web Application Server 3.0.1 "Overview", published Aug. 14, 1998.

Oracle "Developing Your Own Web Application Server™ Cartridge" Release 3.0.1, published Aug. 14, 1998.

Oracle Web Application Server™, "Installation Guide for Sun SPARC Solaris 2.x" Release 3.0.1, published Aug. 14, 1998.

Oracle "Using Oracle Web Application Server™ Cartridge" Release 3.0.1, published Aug. 14, 1998.

Oracle "Performance Tuning", Operating System Parameters (Sun Solaris), published Aug. 14, 1998.

Oracle "Security", "Security Overview", published Aug. 14, 1998.

Oracle Glossary (A–X), published Aug. 14, 1998.

Executive Overview; Oracle Web Application Server™ 3.0; http://www.silexsa.com/oracle/was30 eo.htm; retrieved May 11, 2000.

Oracle Corporation; Oracle WebServer Architecture; Seshu Adunuthula, Mala Anand, Ankur Sharma; http://www.win-.tue.nl/00www/anand.html; dated Apr. 1996; retrieved May 10, 2000.

Distributed Objects on the Internet: Oracle Web Application Server™ 3.0; Richard Delval–Duarte; http://www.fors.com/eoug97/papers/0504.htm; dated Nov. 1996; retrieved May 10, 2000.

Oracle Corporation; WRB API Overview; http://www.cs-.vu.nl/~eliens/WWW5/papers/Broker.html; retrieved.

Web Application Server 3.0 "Oracle Web Application Server Documentation Roadmap".

Oracle Web Application Server™ Installation Guide for Sun SPARC Solaris 2.x, Release 3.0.

Oracle Web Application Server™ Overview, Release 3.0.

Oracle Web Application Server™ Cartridge User's Guide, Release 3.0.

Web Request Broker ™Programmer's Reference, Release 3.0.

Merle, P., et al., "CorbaWeb: A generic object navigator", Computer Networks and ISDN Systems, vol. 28, No. 11, May 1996.

Web Page containing an article written by Rich Levin titled "NetDynamics To Launch Web Database Development System Upgrade," Sep. 29, 1997 (As printed on Dec. 11, 1997).

KIVA Software Corporation, "Developing and Managing Web–based Enterprise Applications".

Modeling transaction integrity: how CASE tools illustrate the relationships between transactions and data; Frank, Maurice, DBMS, v6, n1, p62(5), Jan. 1993.*

Luotonen et al., "World–Wide Web proxies", pp. 147–154, computer Network and ISDN system, 01/94.

James Powell, "Creating a hypertext library information system," pp. 59–66, 02/94.

* cited by examiner

ORCL01622464

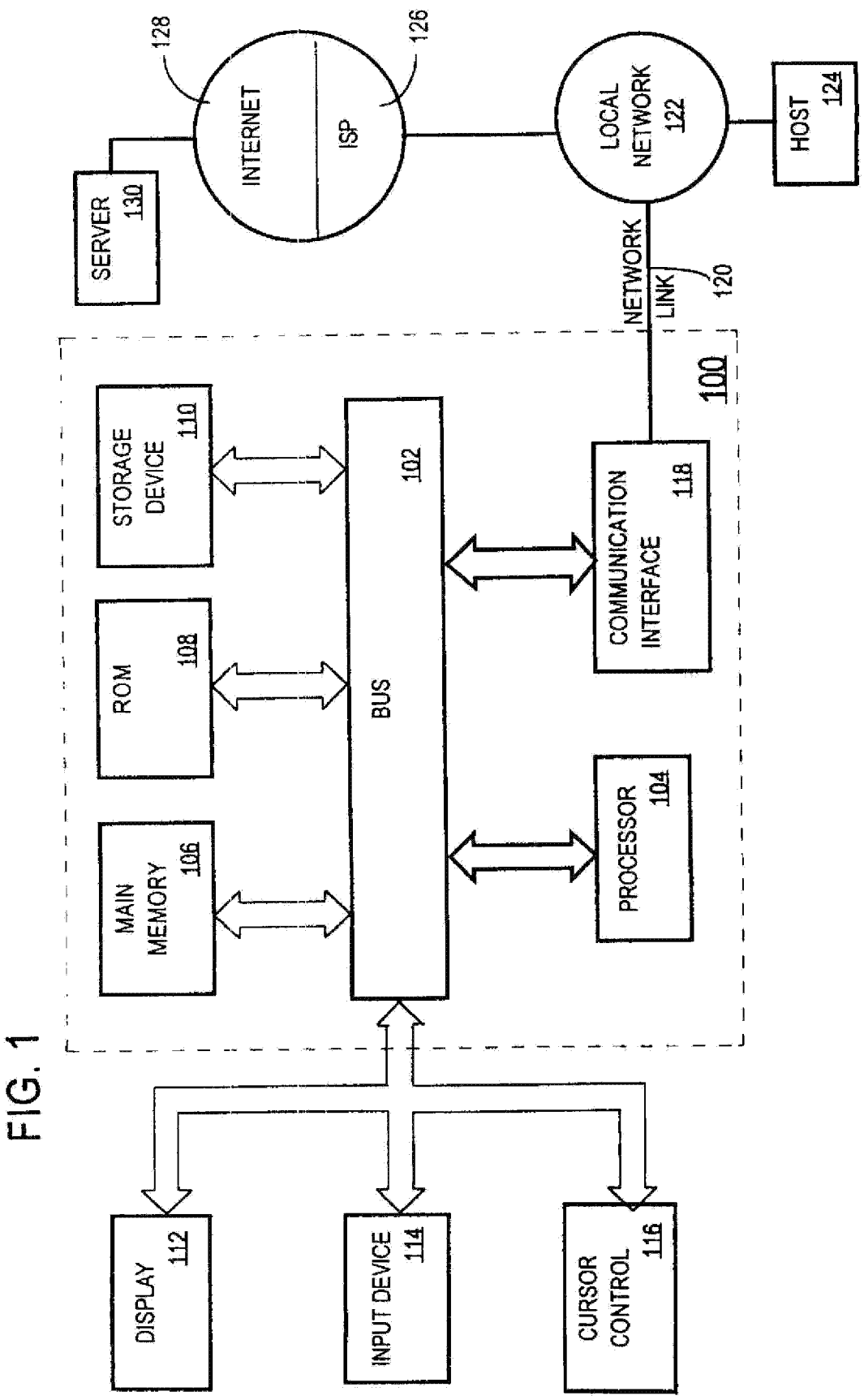

128
126
INTERNET
ISP
LOCAL NETWORK 122
HOST 124
SERVER 130
NETWORK LINK 120
100
STORAGE DEVICE 110
102
COMMUNICATION INTERFACE 118
ROM 108
BUS
PROCESSOR 104
MAIN MEMORY 106
DISPLAY 112
INPUT DEVICE 114
CURSOR CONTROL 116

FIG. 1

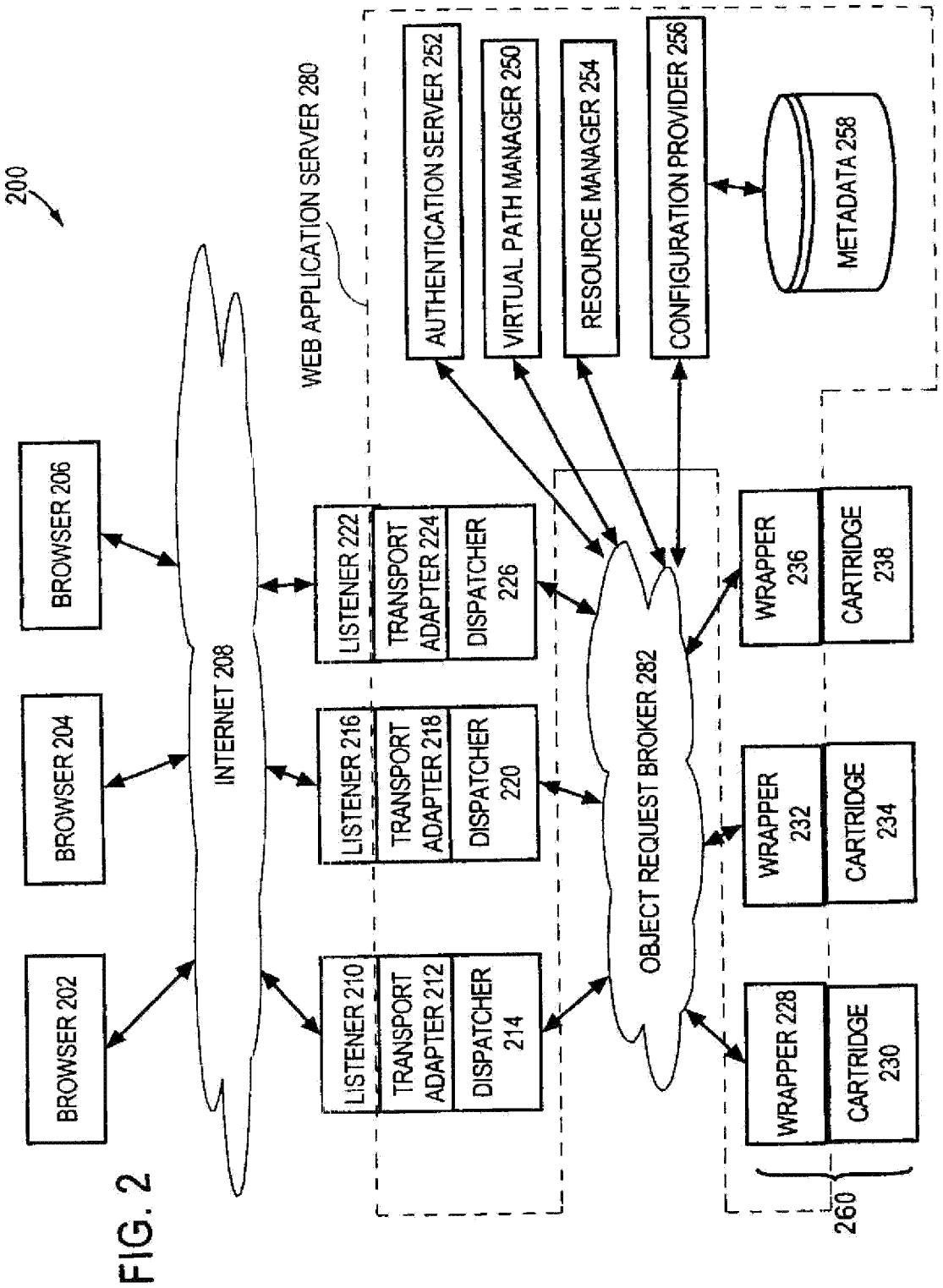

200
WEB APPLICATION SERVER 280
AUTHENTICATION SERVER 252
VIRTUAL PATH MANAGER 250
RESOURCE MANAGER 254
CONFIGURATION PROVIDER 256
METADATA 258
BROWSER 206
BROWSER 204
BROWSER 202
INTERNET 208
LISTENER 222
TRANSPORT ADAPTER 224
DISPATCHER 226
LISTENER 216
TRANSPORT ADAPTER 218
DISPATCHER 220
LISTENER 210
TRANSPORT ADAPTER 212
DISPATCHER 214
OBJECT REQUEST BROKER 282
WRAPPER 236
CARTRIDGE 238
WRAPPER 232
CARTRIDGE 234
WRAPPER 228
CARTRIDGE 230
260

FIG. 2

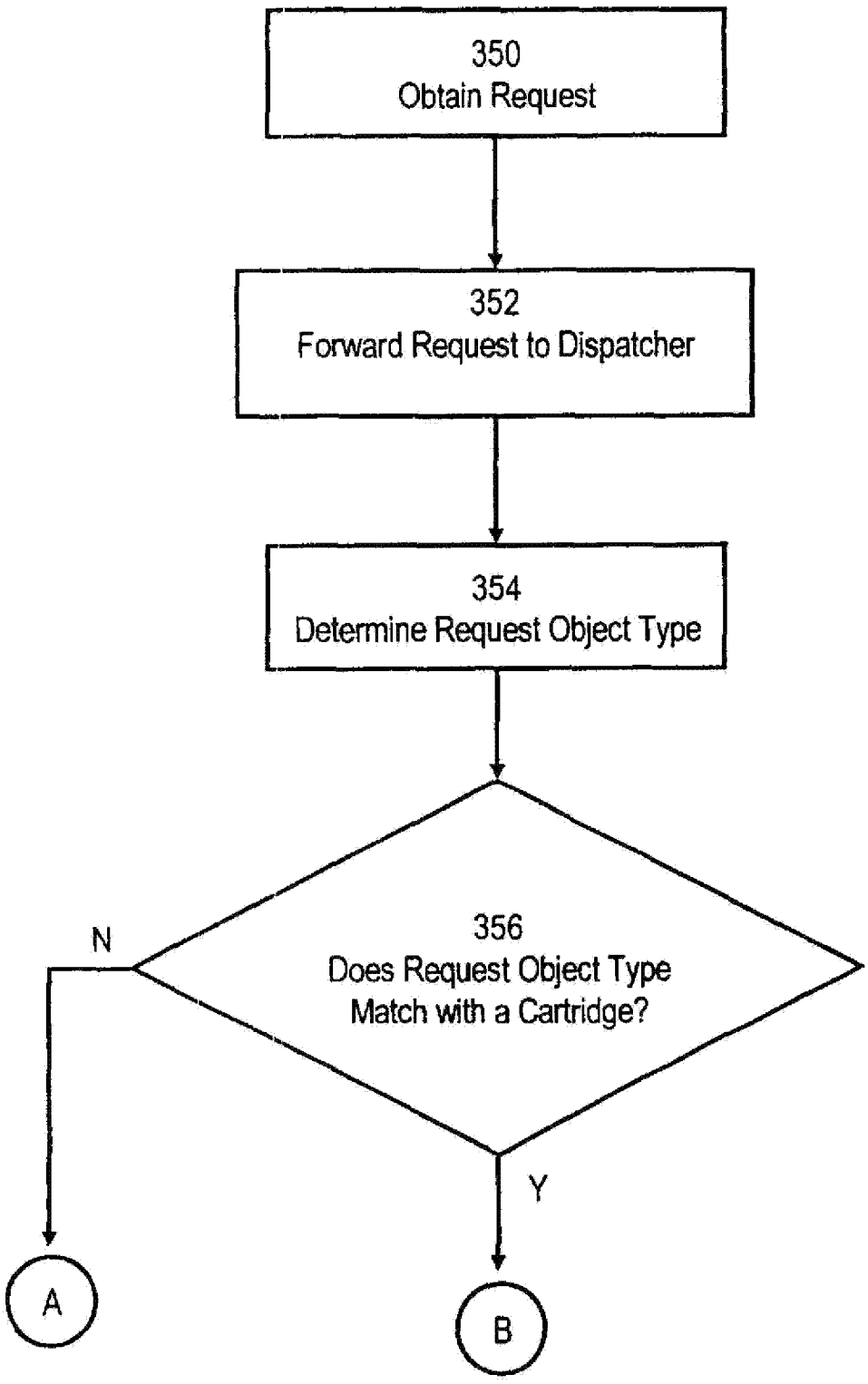
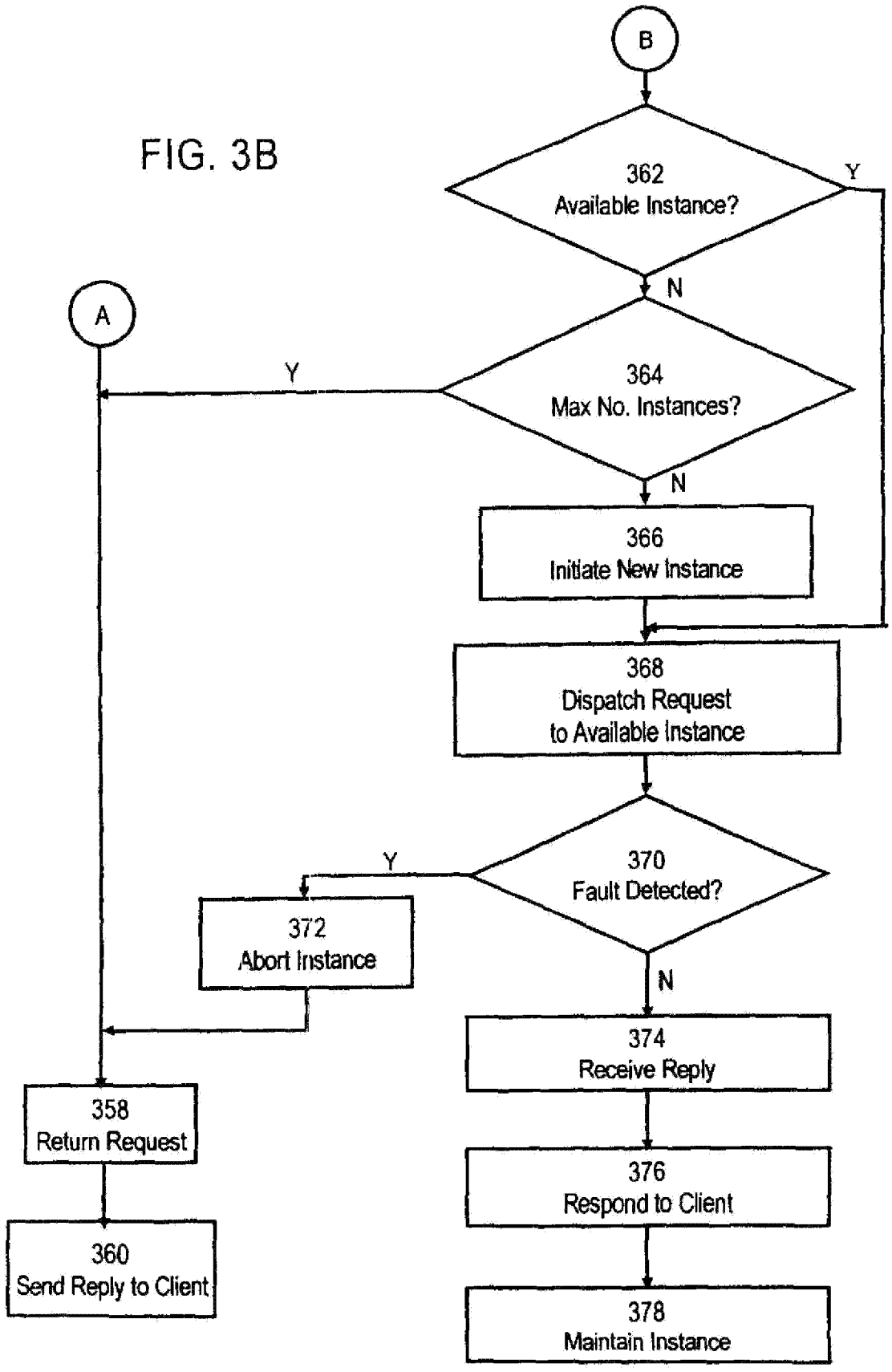

350
Obtain Request
352
Forward Request to Dispatcher
354
Determine Request Object Type
356
Does Request Object Type Match with a Cartridge?
N
Y
A
B

FIG. 3A

FIG. 3B



ORCL01622468

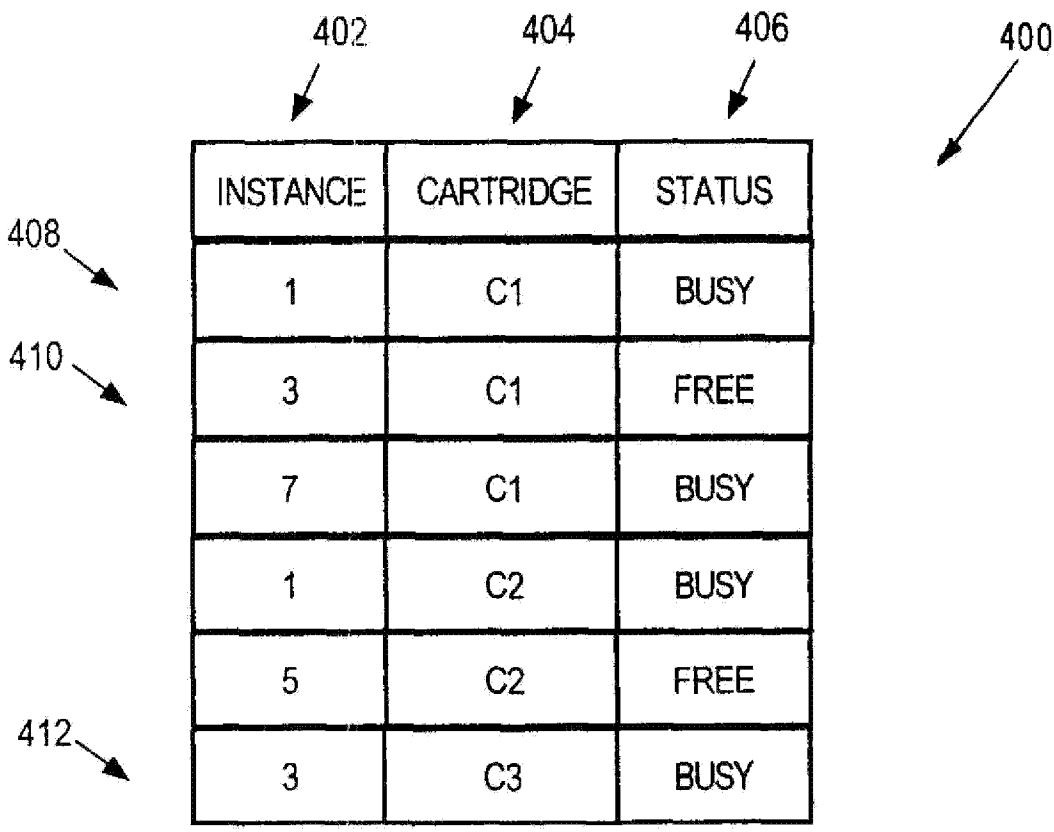

402
404
406
400
INSTANCE
CARTRIDGE
STATUS
408
1
C1
BUSY
410
3
C1
FREE
7
C1
BUSY
1
C2
BUSY
5
C2
FREE
412
3
C3
BUSY

FIG. 4

502 504 506 508

INSTANCE	CARTRIDGE	LISTENER	MACHINE
1	C1	L210	M1
3	C1	L210	M2
7	C1	L210	M3
1	C2	L210	M1
5	C2	L210	M1
3	C3	L210	M3
2	C1	L216	M1
4	C1	L216	M2
5	C1	L216	M3
3	C2	L216	M1
4	C2	L216	M1
1	C3	L216	M3
2	C2	UNOWNED	M1
2	C3	UNOWNED	M2

510 512 500

FIG. 5

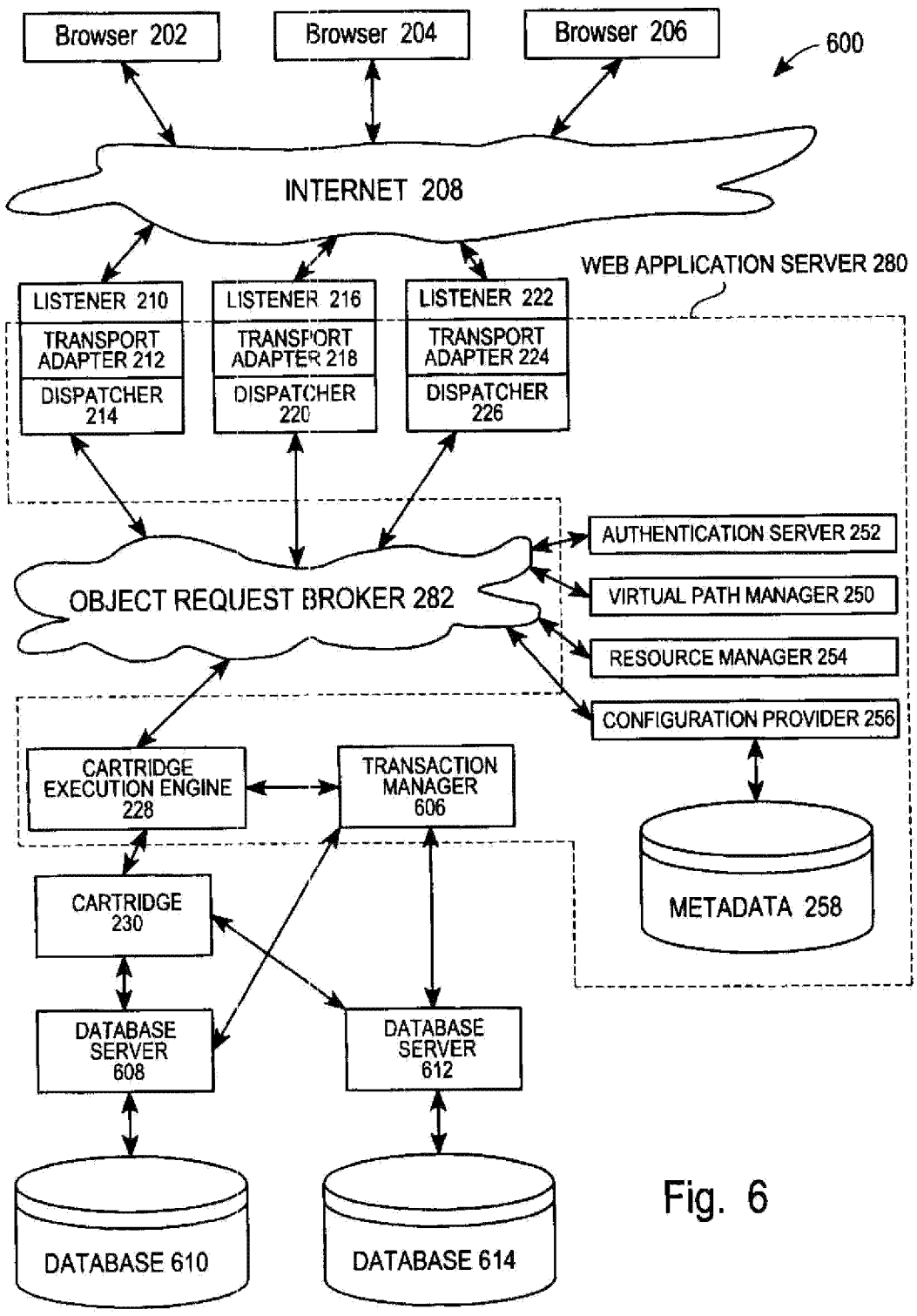

Browser 202
Browser 204
Browser 206
600
INTERNET 208
WEB APPLICATION SERVER 280
LISTENER 210
TRANSPORT ADAPTER 212
DISPATCHER 214
LISTENER 216
TRANSPORT ADAPTER 218
DISPATCHER 220
LISTENER 222
TRANSPORT ADAPTER 224
DISPATCHER 226
AUTHENTICATION SERVER 252
VIRTUAL PATH MANAGER 250
RESOURCE MANAGER 254
CONFIGURATION PROVIDER 256
OBJECT REQUEST BROKER 282
CARTRIDGE EXECUTION ENGINE 228
TRANSACTION MANAGER 606
METADATA 258
CARTRIDGE 230
DATABASE SERVER 608
DATABASE SERVER 612
DATABASE 610
DATABASE 614

Fig. 6

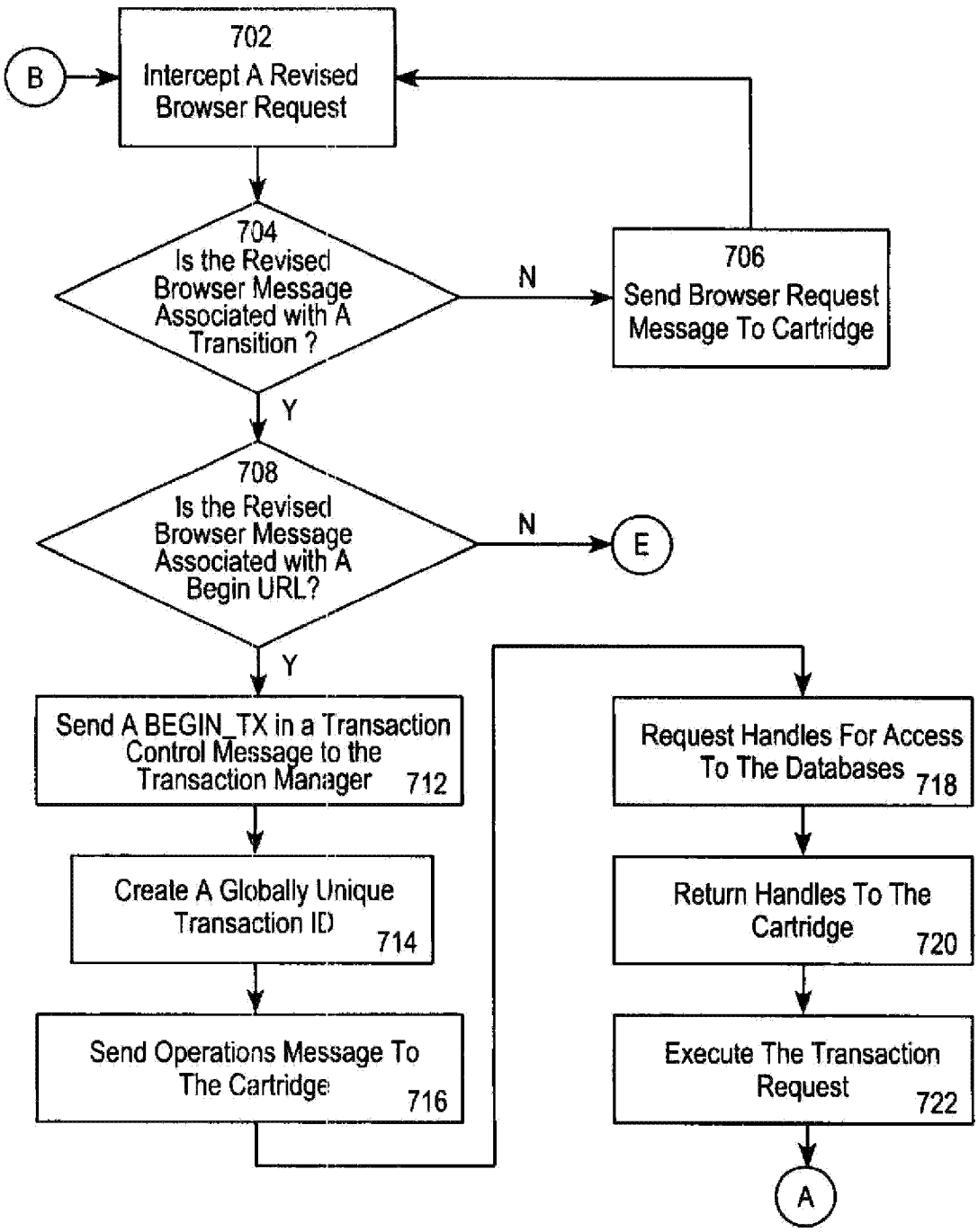

B
702
Intercept A Revised Browser Request
704
Is the Revised Browser Message Associated with A Transition ?
N
706
Send Browser Request Message To Cartridge
Y
708
Is the Revised Browser Message Associated with A Begin URL?
N
E
Y
Send A BEGIN_TX in a Transaction Control Message to the Transaction Manager 712
Create A Globally Unique Transaction ID 714
Send Operations Message To The Cartridge 716
Request Handles For Access To The Databases 718
Return Handles To The Cartridge 720
Execute The Transaction Request 722
A

Fig. 7A

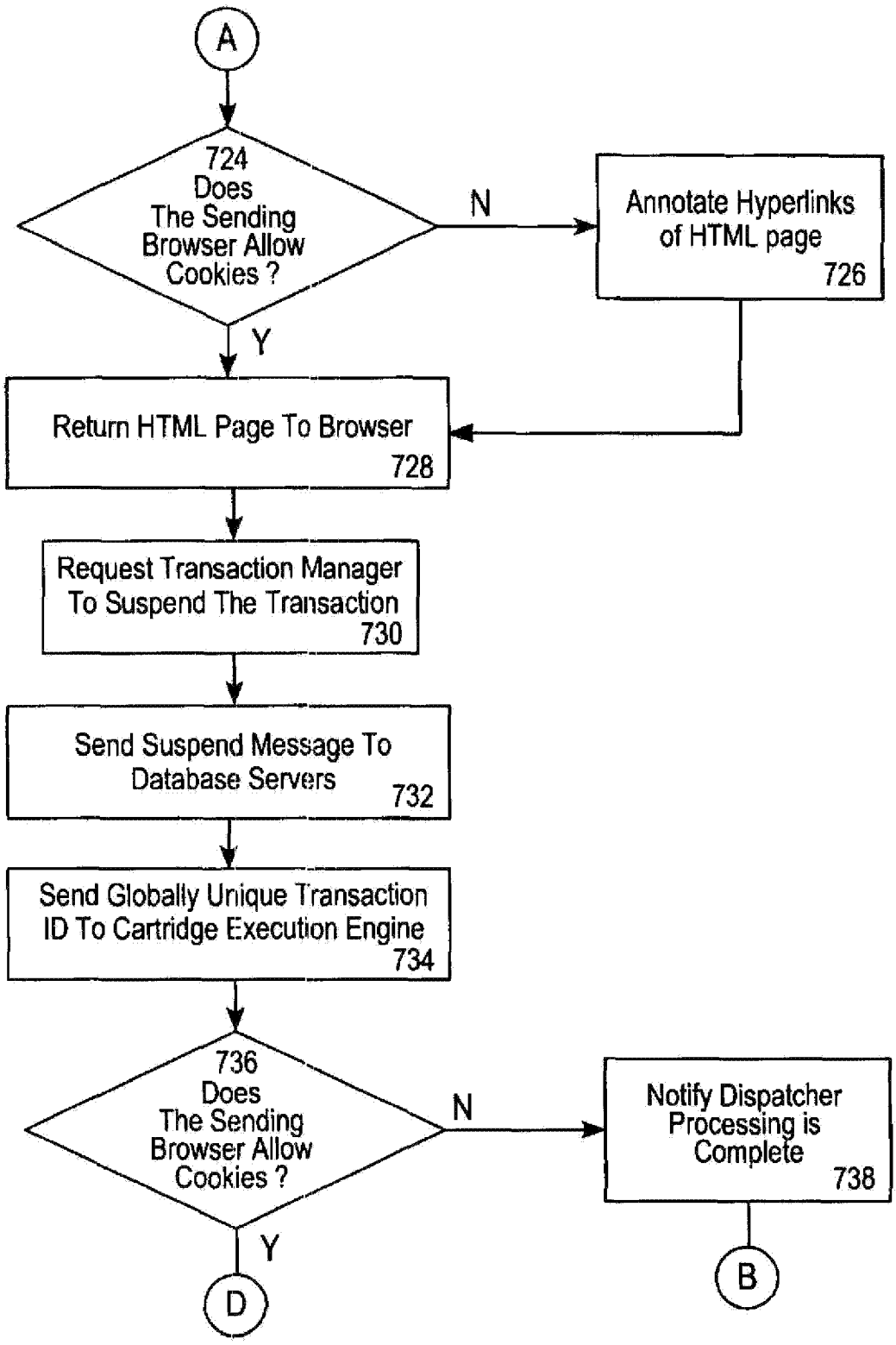


Fig. 7B

ORCL01622473

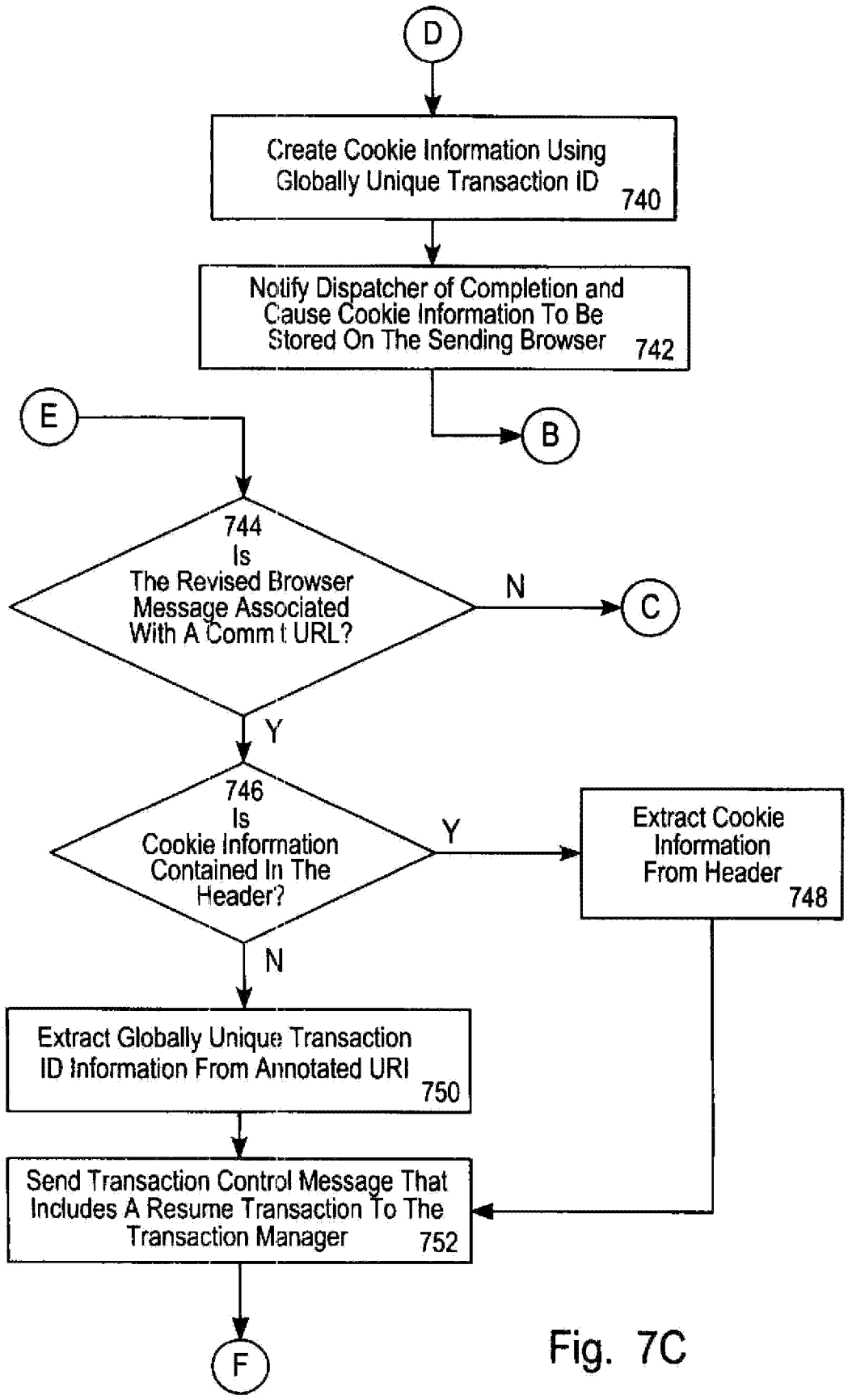

D
Create Cookie Information Using Globally Unique Transaction ID
740
Notify Dispatcher of Completion and Cause Cookie Information To Be Stored On The Sending Browser
742
B
E
744
Is The Revised Browser Message Associated With A Comm t URL?
N
C
Y
746
Is Cookie Information Contained In The Header?
Y
Extract Cookie Information From Header
748
N
Extract Globally Unique Transaction ID Information From Annotated URI
750
Send Transaction Control Message That Includes A Resume Transaction To The Transaction Manager
752
F

Fig. 7C

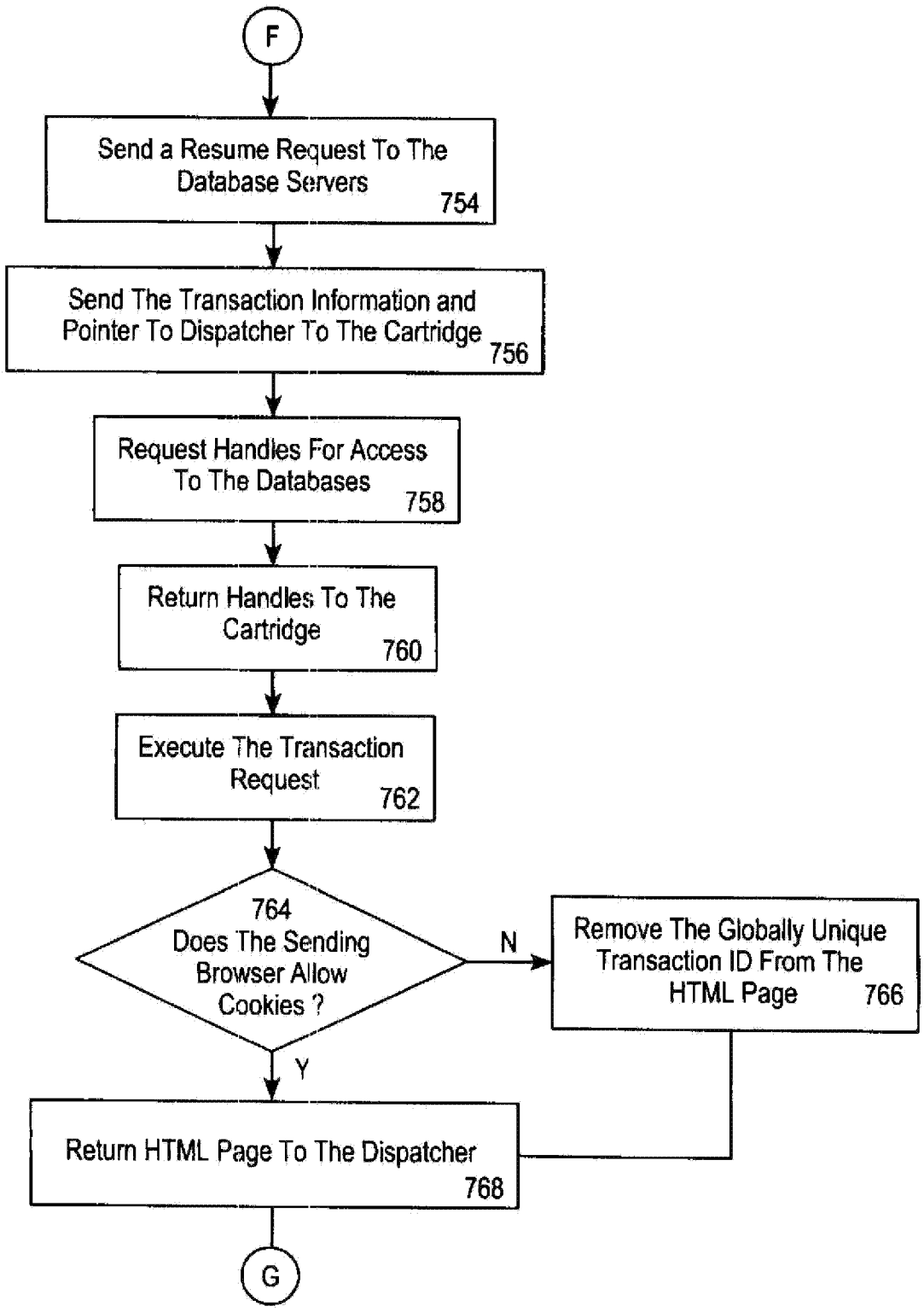

F
Send a Resume Request To The Database Servers
754
Send The Transaction Information and Pointer To Dispatcher To The Cartridge
756
Request Handles For Access To The Databases
758
Return Handles To The Cartridge
760
Execute The Transaction Request
762
764
Does The Sending Browser Allow Cookies ?
N
Remove The Globally Unique Transaction ID From The HTML Page
766
Y
Return HTML Page To The Dispatcher
768
G

FIG. 7D

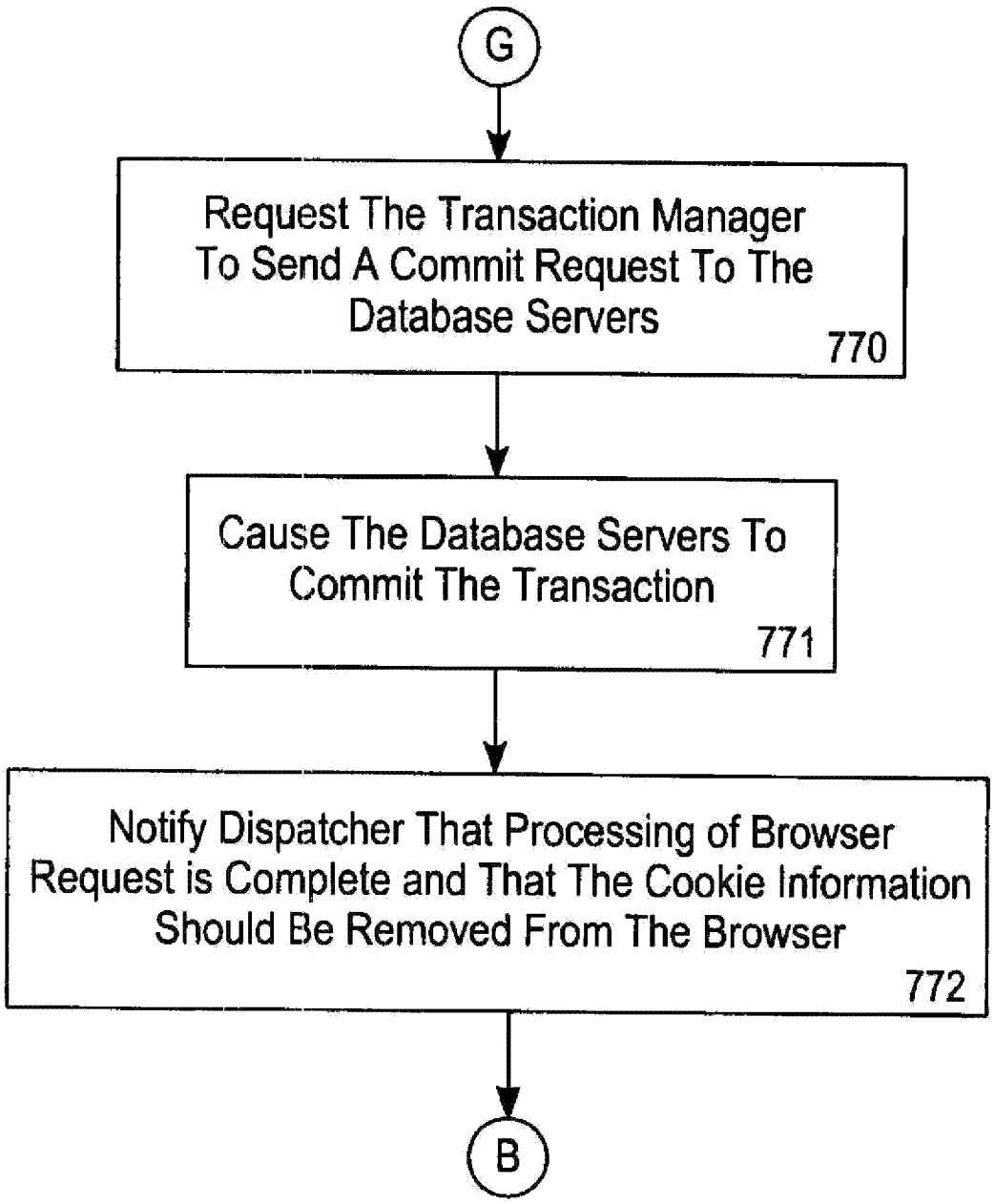

G
Request The Transaction Manager To Send A Commit Request To The Database Servers
770
Cause The Database Servers To Commit The Transaction
771
Notify Dispatcher That Processing of Browser Request is Complete and That The Cookie Information Should Be Removed From The Browser
772
B

Fig. 7E

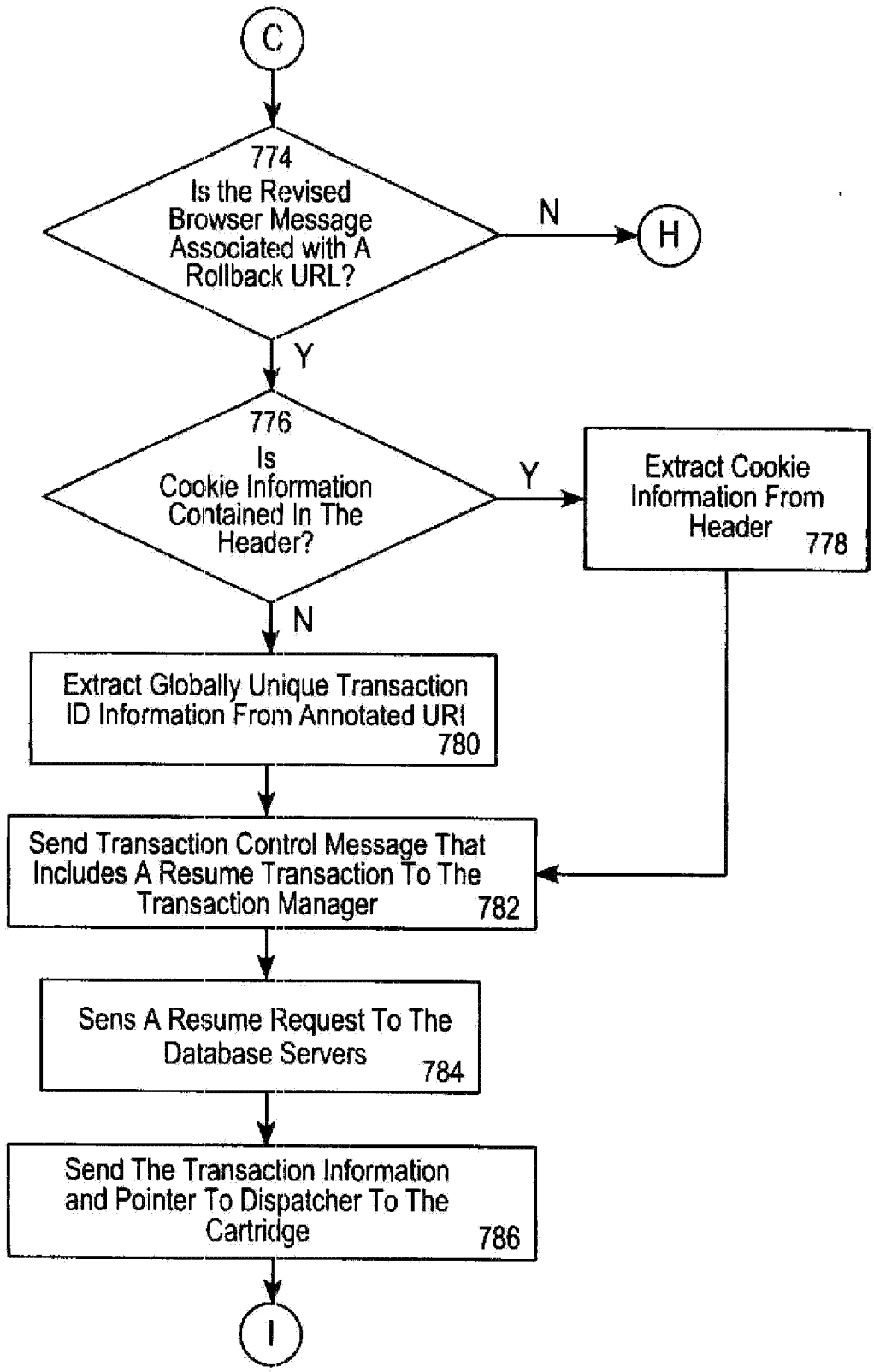

C
774
Is the Revised Browser Message Associated with A Rollback URL?
N
H
Y
776
Is Cookie Information Contained In The Header?
Y
Extract Cookie Information From Header
778
N
Extract Globally Unique Transaction ID Information From Annotated URI
780
Send Transaction Control Message That Includes A Resume Transaction To The Transaction Manager
782
Sens A Resume Request To The Database Servers
784
Send The Transaction Information and Pointer To Dispatcher To The Cartridge
786
I

Fig. 7F

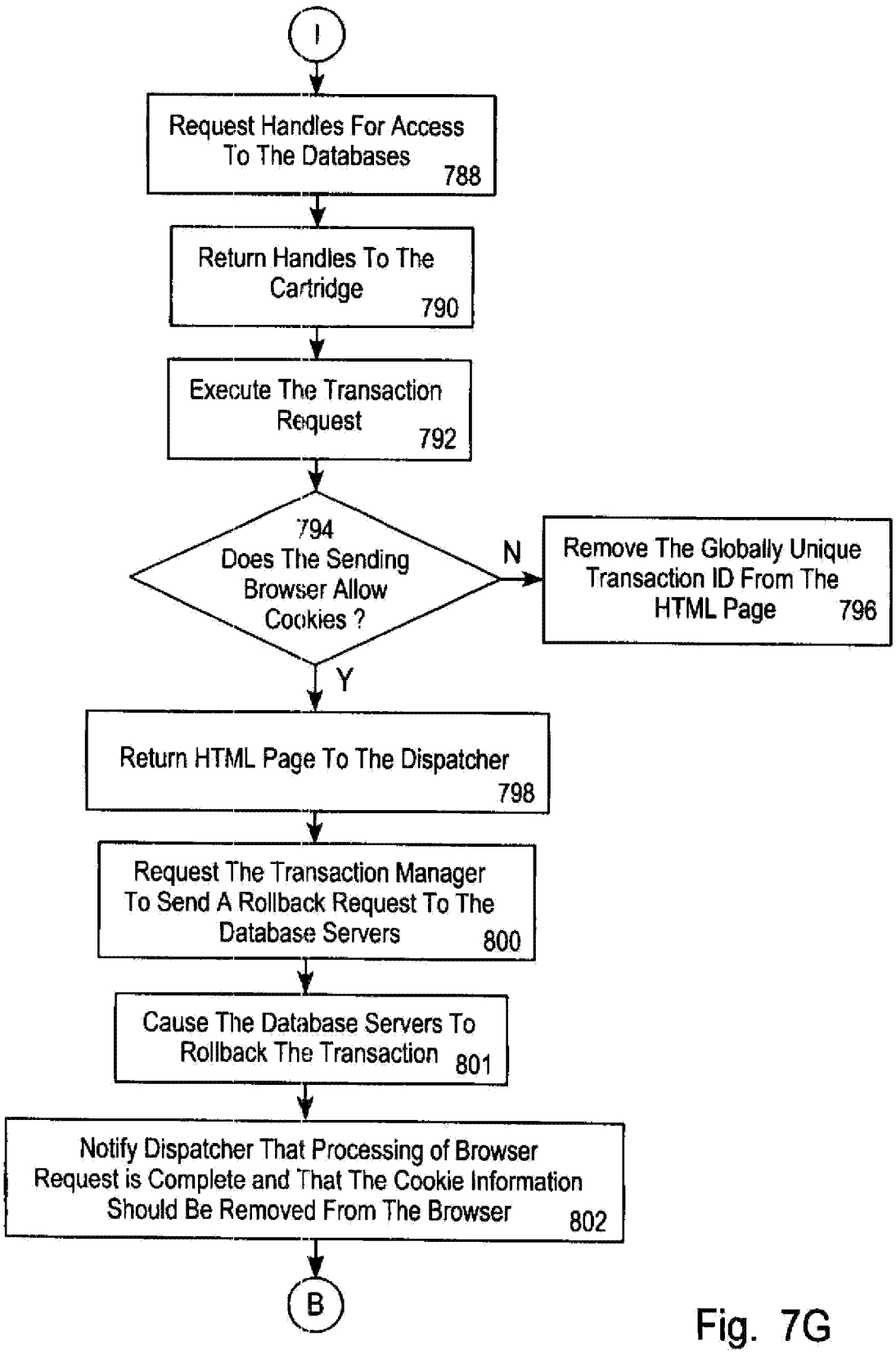

I
Request Handles For Access To The Databases 788
Return Handles To The Cartridge 790
Execute The Transaction Request 792
794 Does The Sending Browser Allow Cookies ?
N
Remove The Globally Unique Transaction ID From The HTML Page 796
Y
Return HTML Page To The Dispatcher 798
Request The Transaction Manager To Send A Rollback Request To The Database Servers 800
Cause The Database Servers To Rollback The Transaction 801
Notify Dispatcher That Processing of Browser Request is Complete and That The Cookie Information Should Be Removed From The Browser 802
B

Fig. 7G

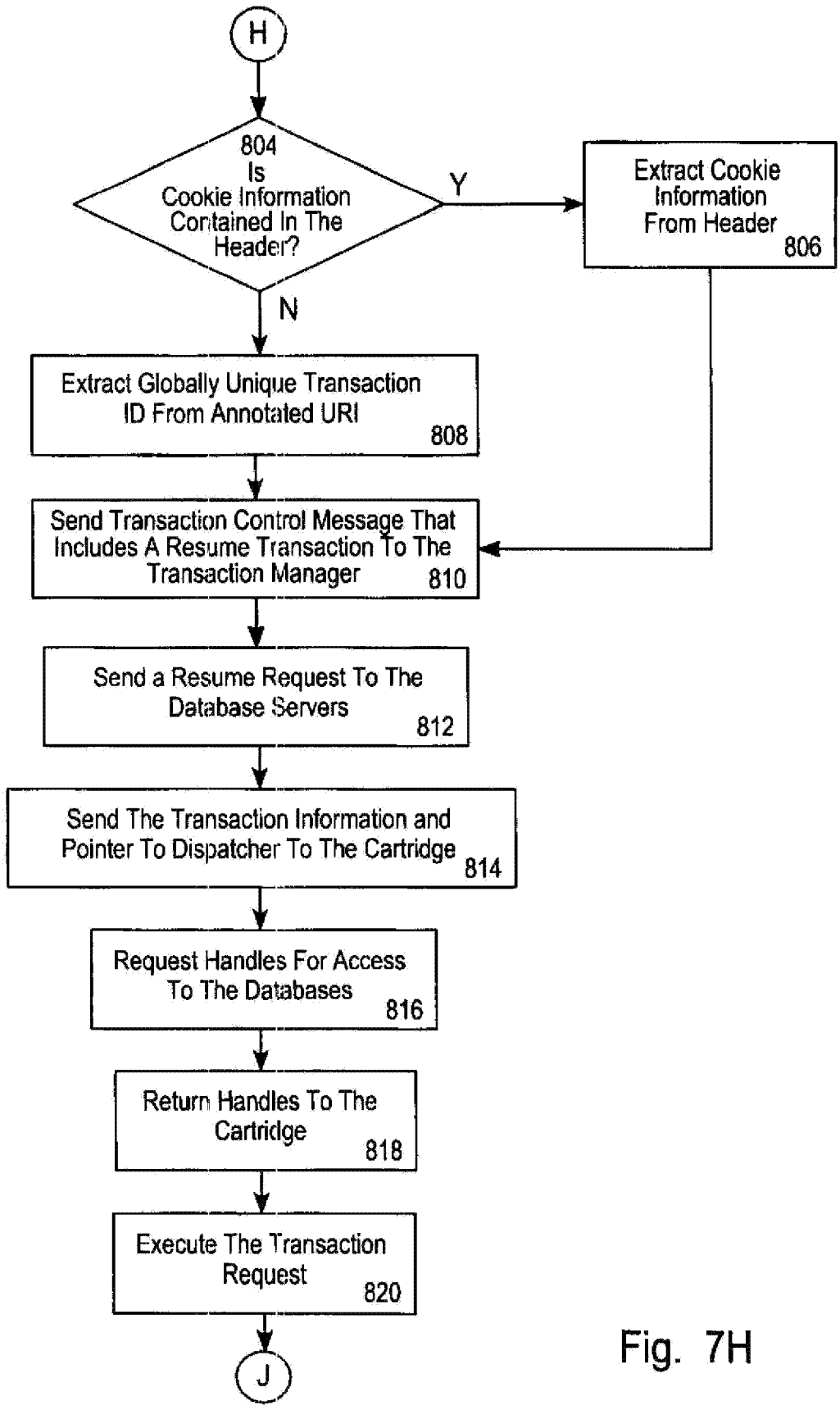

H
804
Is
Cookie Information
Contained In The
Header?
Y
N
Extract Cookie
Information
From Header
806
Extract Globally Unique Transaction
ID From Annotated URI
808
Send Transaction Control Message That
Includes A Resume Transaction To The
Transaction Manager
810
Send a Resume Request To The
Database Servers
812
Send The Transaction Information and
Pointer To Dispatcher To The Cartridge
814
Request Handles For Access
To The Databases
816
Return Handles To The
Cartridge
818
Execute The Transaction
Request
820
J

Fig. 7H

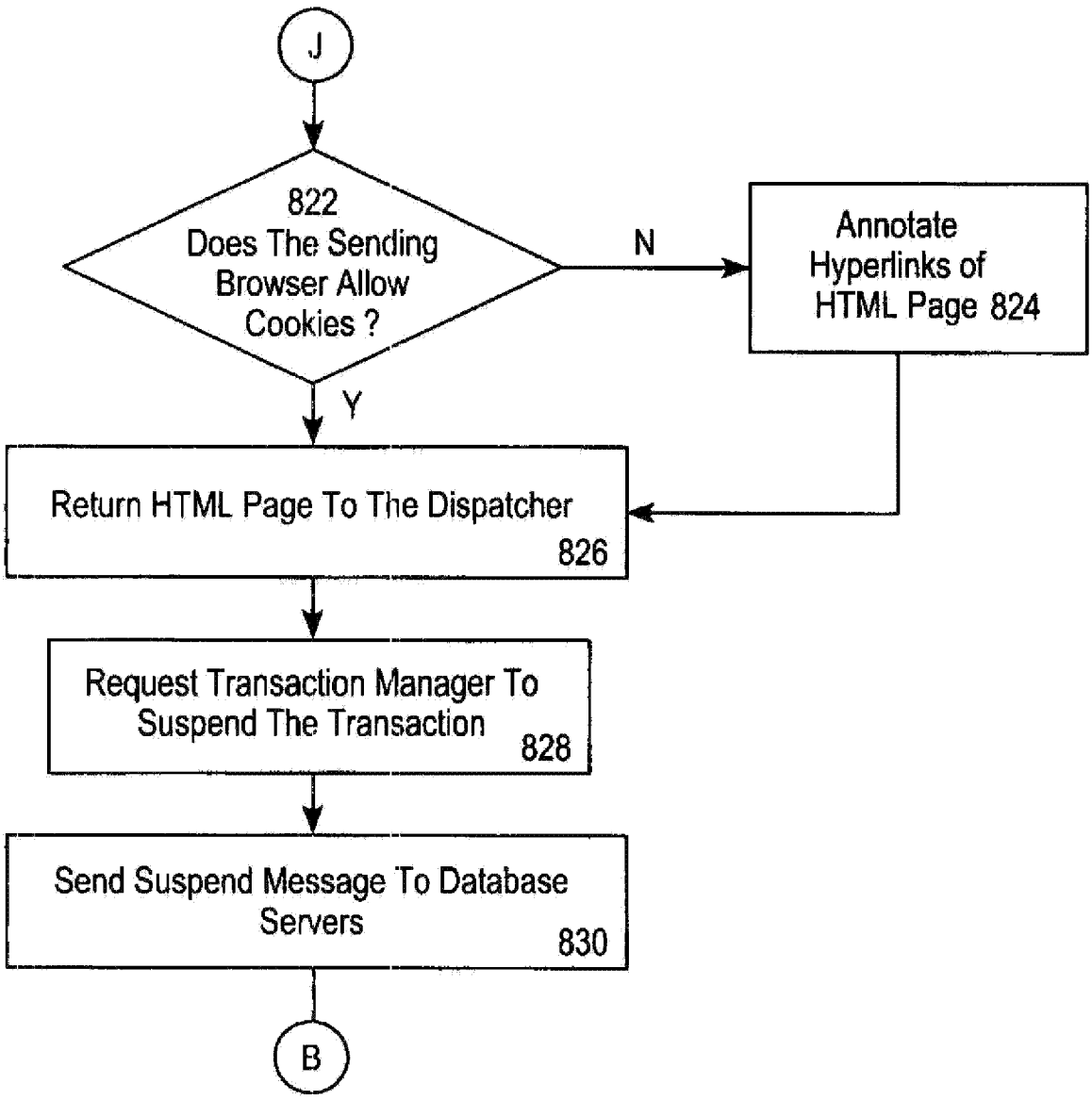

J
822
Does The Sending Browser Allow Cookies ?
N
Annotate Hyperlinks of HTML Page 824
Y
Return HTML Page To The Dispatcher
826
Request Transaction Manager To Suspend The Transaction
828
Send Suspend Message To Database Servers
830
B

FIG. 7I

METHOD AND APPARATUS FOR PERFORMING TRANSACTIONS IN A STATELESS WEB ENVIRONMENT WHICH SUPPORTS A DECLARATIVE PARADIGM

FIELD OF THE INVENTION

This invention relates to processing transactions in networked computer systems, and more specifically to processing multiple-request transactions in a stateless web environment.

BACKGROUND OF THE INVENTION

The World Wide Web includes a network of servers on the Internet, each of which is associated with one or more HTML (Hypertext Markup Language) pages. The HTML pages associated with a server provide information and hypertext links to other documents on that and (usually) other servers. Servers communicate with clients by using the Hypertext Transfer Protocol (HTTP). The servers listen for requests from clients for their HTML pages, and are therefore often referred to as "listeners".

Users of the World Wide Web use a client program, referred to as a browser, to request, decode and display information from listeners. When the user of a browser selects a link on an HTML page, the browser that is displaying the page sends a request over the Internet to the listener associated with the Universal Resource Locator (URL) specified in the link. In response to the request, the listener transmits the requested 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 listeners is in the form of static HTML pages. Static HTML pages are created and stored at the listener 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 listeners that respond to browser requests by performing dynamic operations. For example, a listener 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 listener must be enhanced or augmented. Various approaches have been developed for extending listeners to support dynamic operations.

One of the major characteristics of the web is that it provides a stateless environment. That is, HTTP communicates information on a message-by-message basis without any mechanism for designating relationships between messages. This means that a process servicing a current request cannot determine whether the current request came from the same client as a previous request. In addition, the servicing process cannot determine how or if the current request relates to a previous request.

A disadvantage with using a stateless environment is that it is difficult to process multiple-request transactions. A multiple-request transaction is a set of operations that (1) are specified in more than one request, and (2) must be performed as an atomic unit of work. For example, a multiple-request transaction could consist of three separate operations, such as buying stock item A, selling stock item B and updating the inventory to reflect the number of stock items on hand. Each of these three operations may be specified in a separate request, but each operation should only be performed if all three operations can be performed. In order to properly determine that buying stock item A, selling stock item B and updating the inventory are from the same single transaction requires that transaction state information be retained by the servicing process that receives the three requests.

One possible solution to the stateless problem is to spawn a servicing process for each request-issuing source (each "client"). Each time a request from a client is received, the same servicing process is called upon to process the request. Because the same process is invoked for a given client, the transaction state information for a particular transaction can be maintained by the associated servicing process, thus allowing for the processing of multiple-request transactions.

This solution has significant drawbacks, however. First, maintaining a separate servicing process for each client is wasteful since most clients do not continually make requests to the servicing process. Between client requests, the servicing process simply waits, consuming system resources, without performing any work. A second drawback with this solution is that it is non-scalable. If a servicing process is spawned and maintained for each client, system resources would quickly be consumed, even for a relatively small number of clients. Therefore, spawning a servicing process for each client is not a viable solution for large scale systems.

A second possible solution is to require each servicing process to maintain the current state of the transactions that it is currently processing. By maintaining transaction state information, each servicing process can ensure that multiple-request transactions are processed correctly. However, a drawback associated with requiring each servicing process to maintain transaction state information is that it puts a burden on the developer of each servicing process to write extra code in order to maintain the required transaction state information.

Based on the foregoing, it is desirable to provide a mechanism for processing multiple-request transactions in a stateless environment that does not require a servicing process to maintain transaction state information.

SUMMARY OF THE INVENTION

A method and system for processing multiple-request transactions in a stateless environment is provided.

According to one aspect of the invention, a cartridge execution engine intercepts browser messages directed to a cartridge. The cartridge execution engine determines whether the browser messages are associated with transactions. If the browser messages are associated with transactions, then the cartridge execution engine sends transaction control messages to a transaction manager. The cartridge execution engine also sends operation messages to the cartridge. The cartridge performs the operations specified in the operation messages. In response to the transaction control messages from the cartridge execution engine, the transaction manager causes the multiple-request transactions to be either committed or rolled back as an atomic unit of work.

According to another aspect of the invention, the browser messages associated with transactions are associated with transaction IDs that can be used to identify a browser that is associated with a particular browser message.

According to another aspect of the invention, the browser messages associated with transactions are associated with transaction IDs and are used to identify a browser associated with a particular browser message.

ORCL01622481

According to another aspect of the invention, the transaction IDs are maintained as cookies on the browser that is associated with the particular browser message.

According to another aspect of the invention, the transaction IDs are maintained as URLs on the browser that is associated with the particular browser message.

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 and in which like reference numerals refer to similar elements and in which:

FIG. **1** is a block diagram of a computer system upon which an embodiment of the invention may be implemented;

FIG **2** is a block diagram of a distributed application server according to an embodiment of the invention;

FIG. **3**A is a portion of a flow chart illustrating steps for handling a browser request according to an embodiment of the invention;

FIG. **3**B is another portion of the flow chart illustrating steps for handling a browser request according to an embodiment of the invention;

FIG. **4** is a block diagram of a table containing information maintained by a dispatcher according to an embodiment of the invention;

FIG. **5** is a block diagram of a table containing information maintained by a resource manager according to an embodiment of the invention.

FIG. **6** is a block diagram of a distributed application server for processing transactions according to an embodiment of the invention;

FIG. **7**A is a portion of a flow diagram illustrating steps for processing multiple-request transactions in a stateless environment according to an embodiment of the invention;

FIG. **7**B is another portion of the flow diagram illustrating steps for processing multiple-request transactions in a stateless environment according to an embodiment of the invention,

FIG. **7**C is another portion of the flow diagram illustrating steps for processing multiple-request transactions in a stateless environment according to an embodiment of the invention;

FIG. **7**D is another portion of the flow diagram illustrating steps for processing multiple-request transactions in a stateless environment according to an embodiment of the invention;

FIG. **7**E is another portion of the flow diagram illustrating steps for processing multiple-request transactions in a stateless environment according to an embodiment of the invention;

FIG. **7**F is another portion of the flow diagram illustrating steps for processing multiple-request transactions in a stateless environment according to an embodiment of the invention;

FIG. **7**G is another portion of the flow diagram illustrating steps for processing multiple-request transactions in a stateless environment according to an embodiment of the invention;

FIG. **7**H is another portion of the flow diagram illustrating steps for processing multiple-request transactions in a stateless environment according to an embodiment of the invention; and

FIG. **7**I is another portion of the flow diagram illustrating steps for processing multiple-request transactions in a stateless environment according to an embodiment of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

A method and apparatus for processing multiple-request transactions over a network 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 these 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.

HARDWARE OVERVIEW

FIG. **1** is a block diagram that illustrates a computer system **100** upon which an embodiment of the invention may be implemented. Computer system **100** includes a bus **102** or other communication mechanism for communicating information, and a processor **104** coupled with bus **102** for processing information. Computer system **100** also includes a main memory **106**, such as a random access memory (RAM) or other dynamic storage device, coupled to bus **102** for storing information and instructions to be executed by processor **104**. Main memory **106** also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor **104**. Computer system **100** further includes a read only memory (ROM) **108** or other static storage device coupled to bus **102** for storing static information and instructions for processor **104**. A storage device **110**, such as a magnetic disk or optical disk, is provided and coupled to bus **102** for storing information and instructions.

Computer system **100** may be coupled via bus **102** to a display **112**, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device **114**, including alphanumeric and other keys, is coupled to bus **102** for communicating information and command selections to processor **104**. Another type of user input device is cursor control **116**, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor **104** and for controlling cursor movement on display **112**. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane.

The invention is related to the use of computer system **100** to perform specific operations in response to messages from browsers. According to one embodiment of the invention, the operations are performed by computer system **100** in response to processor **104** executing one or more sequences of one or more instructions contained in main memory **106**. Such instructions may be read into main memory **106** from another computer-readable medium, such as storage device **110**. Execution of the sequences of instructions contained in main memory **106** causes processor **104** to perform the process steps described herein. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the invention. Thus, embodiments of the invention are not limited to any specific combination of hardware circuitry and software.

The term "computer-readable medium" as used herein refers to any medium that participates in providing instructions to processor **104** for execution. Such a medium may take many forms, including but not limited to, non-volatile media, volatile media, and transmission media. Non-volatile media includes, for example, optical or magnetic disks, such

as storage device **110**. Volatile media includes dynamic memory, such as main memory **106**. Transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus **102**. Transmission media can also take the form of acoustic or light waves, such as those generated during radio-wave and infrared data communications.

Common forms of computer-readable media include, for example, a floppy disk, a flexible disk, hard disk, magnetic tape, or any other magnetic medium, a CD-ROM, any other optical medium, punchcards, papertape, any other physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, any other memory chip or cartridge, a carrier wave as described hereinafter, or any other medium from which a computer can read.

Various forms of computer readable media may be involved in carrying one or more sequences of one or more instructions to processor **104** for execution. For example, the instructions may initially be carried on a magnetic disk of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system **100** can receive the data on the telephone line and use an infra-red transmitter to convert the data to an infra-red signal. An infra-red detector coupled to bus **102** can receive the data carried in the infra-red signal and place the data on bus **102**. Bus **102** carries the data to main memory **106**, from which processor **104** retrieves and executes the instructions. The instructions received by main memory **106** may optionally be stored on storage device **110** either before or after execution by processor **104**.

Computer system **100** also includes a communication interface **118** coupled to bus **102**. Communication interface **118** provides a two-way data communication coupling to a network link **120** that is connected to a local network **122**. For example, communication interface **118** may be an integrated services digital network (ISDN) card or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface **118** may be a local area network (LAN) card to provide a data communication connection to a compatible LAN. Wireless links may also be implemented. In any such implementation, communication interface **118** sends and receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

Network link **120** typically provides data communication through one or more networks to other data devices. For example, network link **120** may provide a connection through local network **122** to a host computer **124** or to data equipment operated by an Internet Service Provider (ISP) **126**. ISP **126** in turn provides data communication services through the world wide packet data communication network now commonly referred to as the "Internet" **128**. Local network **122** and Internet **128** both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link **120** and through communication interface **118**, which carry the digital data to and from computer system **100**, are exemplary forms of carrier waves transporting the information.

Computer system **100** can send messages and receive data, including program code, through the network(s), network link **120** and communication interface **118**. In the Internet example, a server **130** might transmit a requested code for an application program through Internet **128**, ISP **126**, local network **122** and communication interface **118**.

The received code may be executed by processor **104** as it is received, and/or stored in storage device **110**, or other non-volatile storage for later execution. In this manner, computer system **100** may obtain application code in the form of a carrier wave.

FUNCTIONAL OVERVIEW OF APPLICATION SERVER

FIG. **2** is a block diagram of a system **200** designed according to an embodiment of the invention. The system **200** includes a plurality of browsers **202**, **204** and **206** that communicate with a plurality of listeners **210**, **216** and **222** over the Internet **208** according to the HTTP protocol. In response to requests from the browsers, the listeners cause a web application server **280** to invoke software modules, referred to herein as cartridges. In the illustrated embodiment, web application server **280** has initiated the execution of three cartridges **230**, **234** and **238**.

The web application server **280** is composed of numerous components, including transport adapters **212**, **218** and **224**, dispatchers **214**, **220** and **226**, an authentication server **252**, a virtual path manager **250**, a resource manager **254**, a configuration provider **256** and a plurality of cartridge execution engines **228**, **232** and **236**. The various components of the web application server **280** shall be described hereafter in greater detail.

Significantly, the numerous components of web application server **280** communicate through an inter-machine communication mechanism, such as an Object Request Broker **282**. Using an inter-machine communication mechanism, cartridge instances that perform the operations specified in browser requests may execute on different machines than the listeners that receive the requests and the browsers that issue the requests. Because the cartridge instances are on different machines than the listeners, the listeners are better insulated against faulty cartridge instances, thus enhancing the reliability and security of the system. In addition, the scalability of the system is greatly increased by spreading the processing burden of executing the cartridge instances among many machines, rather than the same machine that is executing the listener. The ability to distribute cartridge instance execution across multiple machines allows numerous types of load balancing techniques to be used in deciding when and where to spawn new cartridge instances.

A typical operation within system **200** generally includes the following stages:

A browser transmits a request over the Internet **208**.

A listener receives the request and passes it through a transport adapter to a dispatcher.

The dispatcher communicates with virtual path manager **250** to identify a cartridge selected by the browser request and to determine whether the cartridge requires authentication

If the cartridge requires authentication, the dispatcher communicates with the authentication server **252** to determine whether the browser is authorized to access the selected cartridge.

If the authentication server **252** determines that the browser is not authorized to access the selected cartridge, the browser is notified that access has been denied.

However, if access is authorized or the virtual path manager **250** determines that authentication is not required, the dispatcher does one of two things. If the dispatcher knows about an unused instance for that cartridge, the

ORCL01622483

dispatcher sends the request to that instance. If there are no unused cartridge instances for that cartridge, the dispatcher asks the resource manager **254** to create a new cartridge instance. After the instance starts up successfully, the cartridge notifies the resource manager of its existence. The resource manager **254** then notifies the dispatcher of the new instance. The dispatcher creates a revised request based on the browser request and sends the revised request to the new instance.

The cartridge instance handles the revised request and sends a response to the dispatcher.

The dispatcher passes the response back through the listener to the client.

These stages shall be described in greater detail hereafter.

CARTRIDGES

Cartridges are modules of code for performing specific application or system functions. A cartridge forms the basic unit of distribution in the system **200**. According to one embodiment of the invention, cartridges are named using Universal Resource Locators (URLs). Thus, a cartridge name (i.e. URL) has two parts: the IP address of the server on which the cartridge resides, and the virtual path in the server directory structure of the compiled cartridge code. Because cartridges are named using URLs, the cartridge name space is global and cartridges may be accessed using the same messaging techniques as are used to access other web resources, such as documents.

According to one embodiment of the invention, each cartridge has a standard interface which provides a common overall structure for all cartridges. The standard interface defines the interface of routines that are invoked by the web application server **280** under particular conditions. According to one embodiment of the invention, the abstract cartridge interface is as follows:

```
interface Cartridge
{
  boolean inito( );
  boolean authenticate(in Principal user_passwd);
  boolean exec(in Request req _obj, out Response resp
    _obj);
  boolean shutdown( );
}
```

The init( ) routine is responsible for intializing the cartridge instance. This may include invoking the constructors of several subobjects, preforking threads and acquiring all other required shared resources.

The shutdown( ) routine is responsible for cleaning up all of the resources and shutting down the cartridge instance. Once the shutdown( ) routine is invoked on a cartridge instance, it immediately becomes unavailable for servicing subsequent requests.

The authenticate( ) routine validates whether the client requesting the services of the cartridge is authorized to use those services.

The exec( ) routine is the generic way to dispatch all service requests to the cartridge.

EXEMPLARY CARTRIDGES

Each cartridge is either 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 runtime, configured to process database queries according to the Oracle-based Programming Language using Structured Query Language (PL/SQL). The PL/SQL runtime executes a browser request having a database query. The PL/SQL runtime processes the request, for example, by accessing a database server in communication with the cartridge instance via a data link.

Another example of a programmable cartridge is a JAVA runtime interpreter. The JAVA runtime interpreter cartridge enables web application developers to write server-side JAVA applications to process browser requests. Similarly, a custom server may be configured as a cartridge in order to provide dynamic operations such as, for example, accessing processes executed by a third party server.

DISPATCHERS

Dispatchers are software modules configured to route the requests received by listeners to the appropriate cartridges. According to one embodiment of the invention, dispatchers are implemented as server-side program extensions (i.e. "plug-ins"). As such, the dispatchers are loaded into and execute within the same address space as the listeners to which they belong. The dispatchers may be linked with the listener code at compile time or dynamically loaded at runtime.

In the illustrated embodiment, dispatchers **214**, **220** and **226** are associated with listeners **210**, **216** and **222**, respectively. Dispatchers **214**, **220** and **226** selectively route browser requests received by listeners **210**, **216** and 222 to cartridges.

For example, assume that listener **210** receives a browser request over the Internet **208** delivered in the form of a Uniform Resource Locator (URL). The browser request serves as an identifier for a web object, for example an HTML page or an operation to be performed. The listener **210** hands off the browser request to dispatcher **214** without any attempt at interpreting the browser request. Upon receiving the browser request, the dispatcher **214**:

(1) communicates with virtual path manager **250** to identify a cartridge selected by the browser request and to determine whether the cartridge requires authentication,

(2) if the cartridge requires authentication, communicates with the authentication server **252** to determine whether the browser is allowed to access the selected cartridge,

(3) if access is authorized, communicates with the resource manager to determine the specific instance of the selected cartridge to which the browser request should be sent, and

(4) creates and dispatches a revised browser request for execution by the specified instance of the cartridge.

The revised browser request repackages information received in the original browser request. The revised browser request may include, for example, a context object that contains data required for the proper operation of the cartridge. The data required for proper operation of a cartridge may include, for example, a transaction ID that identifies a transaction with which the browser request is associated.

If the cartridge replies to the request, the cartridge sends the reply to the dispatcher and the dispatcher passes the reply up to the listener for transmission to the browser that initiated the request.

CONFIGURATION PROVIDER

According to one embodiment of the invention, cartridges that are to be used with web application server **280** are first

ORCL01622484

registered with web application server **280**. During the registration process, information about the cartridges is supplied to the configuration provider **256**. Configuration provider **256** stores the information as metadata **258** for later access by the components of the web application server **280**.

The metadata **258** may include, for example,

(1) the cartridge name;

(2) the minimum number of required instances;

(3) the maximum number of instances;

(4) the location of the code that implements the cartridge;

(5) the program-dependent function names used by the cartridge execution engine to execute the callback functions (initialization, request handler, shutdown);

(6) a list of machines for running the cartridge;

(7) the idle time for the cartridge (the amount of time instances of the cartridge are allowed to remain idle before they are shut down);

(8) an object identifier; and

(9) data indicating the type of authentication service, if any, to be used with the cartridge.

The object identifier specifies the data that must be supplied by a browser request for requesting performance of an operation by the corresponding cartridge. The object type may be a specific word, a URL, or may include a virtual path such as "/java".

Once the configuration provider **256** has stored the configuration information for a particular cartridge in the metadata **258**, that cartridge is automatically registered when web application server **280** is started.

After a cartridge is registered with the web application server **280**, the resource manager **254** initiates the minimum instances for the cartridge. Once the minimum number of instances has been initiated, the web application server **280** is prepared to process browser requests.

THE VIRTUAL PATH MANAGER

As mentioned above, dispatchers communicate with the virtual path manager **250** to determine where to route each revised browser request. Specifically, each browser request typically includes a URL. Upon receiving a browser request, the dispatcher sends the URL in the request to the virtual path manager **250**. The virtual path manager **250** responds by sending the dispatcher data that identifies the cartridge, if any, associated with the URL.

In order to supply the required information to dispatchers, virtual path manager **250** consults the metadata **258** that maps URLs to cartridges. In response to receiving a browser request, the virtual path manager **250** uses the mapping data to determine the cartridge, if any, to which the URL contained in the browser requests corresponds

For example, if the browser request is a URL request beginning with the virtual path "/java", the mapping may indicate that the JAVA interpreter cartridge is configured to handle requests having the virtual path "/java".

According to one embodiment of the invention, the virtual path manager **250** also determines whether the cartridge associated with the URL requires authentication. If the cartridge requires authentication, the virtual path manager **250** indicates in the response that the virtual path manager **250** sends to the dispatcher that authentication is required. If authentication is not required, the dispatcher creates and sends a revised browser request to an instance of the cartridge without invoking the authentication server **252**. If authentication is required, the dispatcher sends the revised request to an instance of the cartridge only after the authentication server indicates that the revised request may be submitted to an instance of the cartridge.

THE RESOURCE MANAGER

The resource manager **254** of the web application server **280** manages the execution of each of the cartridges by initiating a predetermined minimum number of instances for the cartridges, load balancing between the instances of each cartridge, and initiating new instances of cartridges as necessary up to a predetermined maximum number of instances of a given cartridge.

For example, assume that the metadata for a particular cartridge (C1) includes the following information:

Name=C1
Minimum Instances=10
Maximum Instances=50
Host Machines =M1, M2, M3
Idle time =30 seconds

Based on this metadata, when cartridge **C1** is first registered, resource manager **254** will initiate ten instances of C1. Resource manager **254** will initiate the ten instances on the machines associated with the labels **M1, M2** and **M3**.

Upon receipt of requests from dispatchers to access C1, resource manager **254** determines whether any existing instance of C1 is available for use. If no instance of C1 is available when a request is received, resource manager **254** determines whether the maximum number of instances of C1 are already running. If the maximum number of instances of C1 are not already running, then resource manager **254** initiates a new instance of C1 on one of the possible host machines and transmits a message that identifies the new instance to the dispatcher that issued the request. If the maximum number of instances of C1 are already running, then resource manager **254** sends a message to the dispatcher that issued the request to indicate that the request cannot be handled at this time.

LOAD BALANCING

According to one embodiment of the invention, resource manager **254** applies a set of load balancing rules to determine where to initiate instances of cartridges where there is more than one possible host machine. Thus, in the above example, **M1, M2** and **M3** are all capable of executing instances of cartridge C1. If **M1, M2** and **M3** have the same processing capacity, it may be desirable to distribute the instances evenly across the three machines. However, if M1 has ten times the processing power of **M2** and **M3**, it may be desirable to initiate all instances of **C1** on **M1** up to a certain point, and then to distribute additional instances evenly among **M1, M2** and **M3**.

To assist resource manager **254** in determining how to load balance among possible machines, the metadata stored for each cartridge may include additional details. For example, the metadata may specify a separate minimum and maximum number of instances for each machine. Resource manager **254** may then distribute new instances among the machines based on which machine has the lowest ratio of actual instances to maximum instances.

The metadata may also specify an order for the machines that can run a cartridge. The machine at the N+1 position in the order is only used to execute instances of the cartridge when the machine at the Nth position in the order is already executing its maximum number of instances.

CARTRIDGE INSTANCE STATUS TRACKING

According to one embodiment of the invention, the resource manager **254** maintains state information to keep

ORCL01622485

track of cartridge instances that have been created. The state information includes data that identifies the instance, identifies the machine executing the instance, and identifies the listener to which the instance has been assigned.

FIG. **5** illustrates a table **500** that may be maintained by resource manager **254** to store this state information. Table **500** includes an instance column **502**, a cartridge column **504**, a listener column **506** and a machine column **508**. Each row of table **500** corresponds to a distinct cartridge instance. Within the row for a given cartridge instance, cartridge column **504** identifies the cartridge associated with the cartridge instance and instance column **502** indicates the instance number of the cartridge instance. For example, row **510** corresponds to an instance of cartridge C1. Therefore, cartridge column **504** of row **510** indicates cartridge C1. Instance column **502** of row **510** indicates that the cartridge instance associated with row **510** is instance **1** of cartridge C1.

Listener column **506** indicates the listener to which the cartridge instance associated with a row has been assigned. Machine column **508** indicates the machine on which the cartridge instance associated with a row is executing. For example, the cartridge instance associated with row **510** has been assigned to listener **210** and is executing on machine M1.

Similar to resource manager **254**, each dispatcher maintains state information for the cartridge instances that have been assigned to the listener to which the dispatcher is attached. Such state information may be maintained, for example, in a table **400** as shown in FIG. **4**. Similar to table **500**, table **400** includes an instance column **402** and a cartridge column **404** that respectively hold instance numbers and cartridge identifiers. However, while table **500** includes one entry for every cartridge instance assigned by resource manager **254**, table **400** only includes entries for cartridge instances that have been assigned to a particular listener. For example, table **400** includes entries for only those cartridge instances listed in table **500** that have been assigned to listener **210**.

In addition to instance column **402** and cartridge column **404**, table **400** includes a status column **406**. For each row, the status column **406** holds a value that indicates the status of the instance associated with the row. For example, the status column **406** of row **408** indicates that instance **1** of cartridge C1 is currently busy. In the illustrated embodiment, the status column **406** holds a flag that indicates that a cartridge instance is either BUSY or FREE. The significance of the cartridge status shall now be describe with reference to the operation of resource manager **254** and dispatchers **214** and **220**.

INTERACTION BETWEEN DISPATCHERS AND THE RESOURCE MANAGER

As explained above, dispatchers communicate with resource manager **254** when they need to send a revised browser request to a particular cartridge. According to one embodiment of the invention, dispatchers first determine whether an instance of the appropriate cartridge (1) has already been assigned to it and (2) is available to process the new revised browser request. If an appropriate cartridge instance has already been assigned to the dispatcher and is currently available to process the new revised browser request, then the dispatcher forwards the revised browser request to the cartridge instance without further communication with resource manager **254**.

For example, assume that listener **210** receives a browser request that, according to virtual path manager **250**, must be processed by cartridge C1. Assume also that table **400** reflects the current list and status of cartridge instances that have been assigned to listener **210**. Upon receiving the browser request from listener **210**, dispatcher **214** inspects table **400** to locate a FREE instance of cartridge C1. In the illustrated table **400**, row **410** indicates that instance **3** of cartridge C1 is currently FREE. Consequently, dispatcher **214** forwards a revised browser request directly to instance **3** of cartridge C1 without further communication with resource manager **254**. In response to sending the revised browser request, dispatcher **214** changes the status value in status column **406** of row **410** to BUSY.

If a listener has not already been assigned an appropriate cartridge instance that is currently available, then the dispatcher associated with the cartridge requests a cartridge instance from the resource manager **254**. If the resource manager **254** determines that an instance of the required cartridge is not available and the number of existing instances of the required cartridge is below the maximum, then the resource manager **254** initiates a new cartridge. Upon initiating a new cartridge, the resource manager **254** inserts an entry for the new cartridge instance in table **500**.

Assume, for example, that listener **210** receives a browser request that must be processed by cartridge C3. Assume also that instance **3** of cartridge C3 has not yet been initiated. Under these conditions, dispatcher **214** sends to resource manager **254** a request for a handle to an instance of cartridge C3. In response to this request, resource manager **254** initiates instance **3** of cartridge C3 on machine M3. In addition, resource manager **254** inserts into table **500** the entry found at row **512**.

After inserting row **512** for instance **3** of cartridge C3 in table **500**, resource manager **254** sends back to the dispatcher **214** a handle to the newly created instance. In response to receiving this handle, dispatcher **214** inserts an entry (row **412**) for the new instance in its status table **400**. The dispatcher **214** then transmits a revised browser request to instance **3** of cartridge C3.

RELEASING CARTRIDGE INSTANCES

According to one embodiment of the invention, listeners do not automatically release ownership of cartridge instances when the cartridge instances finish responding to outstanding browser requests. For example, assume that instance **3** of cartridge C3 receives a revised browser request, processes the revised browser request, and sends a response back to dispatcher **214**. Dispatcher **214** passes the response to listener **210** to be sent back to the browser that issued the browser request.

At this point, listener **210** no longer requires ownership of instance **3** of cartridge C3. However, rather than transferring ownership of instance **3** of cartridge C3 back to resource manager **254**, dispatcher **214** merely changes the status column **406** of row **412** from BUSY to FREE.

Changing the value in status column **406** of row **412** to FREE indicates that instance **3** of cartridge C3 is no longer working on a request, and is therefore ready to handle subsequent requests. However, because table **400**, which indicates that instance **3** of cartridge C3 is available, is maintained locally by dispatcher **214**, instance **3** of cartridge C3 is only available for subsequent browser requests arriving at listener **210**. Row **512** of table **500** maintained by resource manager **254** continues to indicate that instance **3** of cartridge C3 is owned by listener **210**.

Because listeners do not automatically release cartridge instances every time a request is serviced, overhead associ-

ORCL01622486

ated with communication between the resource manager **254** and the various dispatchers is significantly reduced. For example, assume that a listener **210** receives ten successive requests that must be communicated to cartridge C3. Rather than communicating with resource manager **254** for each of the ten requests, dispatcher **214** may communicate with resource manager **254** in response to the first request. The subsequent nine requests can be handled by dispatcher **214** without communicating with resource manager **254** because the dispatcher **214** uses the same instance of C3 that processes the first request to process the nine subsequent requests.

While not automatically releasing listener ownership of cartridge instances when each request is serviced can increase the efficiency of web application server **280**, listeners cannot maintain ownership of cartridge instances indefinitely. For example, instances that have not been used for long periods of time should be passed back to the resource manager **254** so they can be de-allocated to free up resources. In addition, it is not efficient for one listener to maintain ownership of the instance of a cartridge that it has not used for a relatively long time when other listeners require instances of that cartridge.

Consequently, resource manager **254** communicates to each listener a maximum idle time for each cartridge instance passed to the listener. The maximum idle time indicates the maximum amount of time a cartridge instance can go unused before the listener must release ownership of the cartridge instance. For example, assume that the resource manager **254** indicates to listener **210** that the maximum amount of idle time for instance 3 of cartridge C3 is 10 minutes. Based on this information, listener **210** may continue to use instance **3** of cartridge C3 to process browser requests for cartridge C3 as long as instance **3** of cartridge C3 does not remain idle or FREE for more than 10 minutes.

If instance **3** of cartridge C3 is idle for more than 10 minutes, dispatcher **214** removes row **412** from table **400** and sends a message to resource manager **254** that listener **210** is releasing ownership of instance **3** of cartridge C3. In response to this message, resource manager **254** updates row **512** to indicate that instance 3 of cartridge C3 is not owned by any listener and may thus be reassigned to another listener or terminated.

In an alternative embodiment, dispatchers do not automatically release cartridge instances when the idle time for the cartridge instance has expired. Instead, the dispatcher sends a message to resource manager **254** offering to release the expired instance. Resource manager **254** may respond to the offer by requesting that the listener release the cartridge instance, or by allowing the listener to retain ownership of the expired cartridge instance.

According to one embodiment of the invention, resource manager **254** maintains a queue of the requests that cannot be immediately serviced. When it becomes possible to service a queued request, the request is removed from the queue and processed.

For example, assume that listener **222** receives a browser request that must be processed by cartridge C1, and that listener **222** has not been assigned any instances of cartridge C1. Dispatcher **226** sends a request for an instance of C1 to resource manager **254**. Assume further that a maximum of **50** instances of C1 are allowed, and that 50 instances of C1 have been assigned to listener **210**. Under these conditions, resource manager **254** cannot service the request from listener **222**. Therefore, resource manager **254** puts the request on a queue. When listener **210** releases an instance of C1, resource manager **254** communicates to listener **222** that an instance of C1 is available.

Under certain conditions, resource manager **254** may preemptively cause a listener to release a cartridge instance. For example, resource manager **254** may detect a system overload situation and respond by terminating a set of cartridge instances, either before or after informing the listeners that currently have been assigned the cartridge instances that the cartridge instances are going to be terminated.

Resource manager **254** may also preemptively cause listeners to release cartridge instances to implement fairness policies between listeners. For example, resource manager **254** may cause a listener that holds the most instances of a given cartridge to release an instance of the cartridge when another listener has waited more than a predetermined threshold of amount of time for an instance of the cartridge. For example, if listener **210** has been assigned **50** instances of cartridge C1 and C1 has a maximum of **50** instances, then resource manager **254** may cause listener **210** to release an instance of C1 ten seconds after receiving a request for an instance of C1 from another listener.

CARTRIDGE EXECUTION ENGINES

According to one embodiment of the invention, each cartridge instance is composed of a cartridge execution engine and a cartridge. A cartridge execution engine is a code module that insulates cartridges from the complexities of the web application server **280** and the inter-module communication mechanism. A cartridge is made available to a cartridge execution engine by storing in a function table pointers to the cartridge functions. According to one embodiment, all cartridges provide the functions specified in the exemplary cartridge interface described above. By having all cartridges support the same interface, a single standard cartridge execution engine can be used with all cartridges.

According to one embodiment of the invention, cartridges are implemented as shared libraries, and cartridge execution engines are executable programs that invoke the routines in the shared libraries using the standard cartridge interface. The cartridge execution engine provides the interface between cartridges and the dispatcher, directs cartridge flow of control, and provides services for cartridges to use.

When the resource manager **254** requires the creation of a new cartridge instance, the resource manager **254** causes a cartridge execution engine to be instantiated. In turn, the instance of the cartridge execution engine thus created causes the appropriate cartridge to be instantiated. The resource manager **254** can cause the cartridge execution engine to be instantiated, for example, by invoking a "cartridge execution engine factory" that resides on the machine on which the cartridge is to be executed. The instance of the cartridge execution engine can cause the cartridge to be instantiated, for example, by making a call to one of the routines in the shared library that constitutes the cartridge.

As shown in FIG. **2**, the web application server **280** includes cartridge execution engines **228**, **232** and **236** for each of the cartridges **230**, **234** and **238**. The cartridge execution engines control execution of the instances of the corresponding cartridges by making calls into the cartridges through the standard cartridge interface. By establishing basic callback functions between the cartridge execution engine and a cartridge, any cartridge can be integrated into the web application server **280** by configuring the cartridge to respond to the callback functions, and then registering the cartridge in the configuration provider **256**, as described below.

ORCL01622487

Thus, if the dispatcher **214** determines that the PL/SQL runtime cartridge is the appropriate cartridge to process a request, the dispatcher **214** dispatches the request to a cartridge instance that includes a cartridge execution engine associated with the PL/SQL runtime cartridge. If a new instance needs to be initiated, the resource manager **254** creates a new instance of the PL/SQL runtime cartridge in a separate address space and dispatches the request to the cartridge execution engine **228** 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 one or more of the components of web application server **280** is executing, or on another computer system.

In response to a message from a dispatcher, the cartridge execution engine issues a request handler callback function to the cartridge, causing the cartridge to process the request. The cartridge executing the request returns the result to the cartridge execution engine, which forwards the result to the dispatcher. In the event that the web application server **280** detects a fault in the operation, the cartridge execution engine issues a shutdown function of the cartridge.

Hence, the cartridge execution engine provides an application programming interface to the web application server **280** that specifies predetermined operations to be performed. Use of the standard cartridge interface enables programmers of the cartridges to configure each cartridge for high-level integration into the web application server **280** independent of the protocols used by the particular web listener with which the cartridge will be used.

TRANSPORT ADAPTERS

Listeners enable the use of server-side plug-ins by providing a programming interface and protocol for use by such plug-ins. Unfortunately, the programming interfaces and protocols provided by listeners vary from listener to listener. For example, Netscape Server Application Programming Interface (NSAPI), Internet Server Application Programming Interface (ISAPI) and Application Development Interface (ADI) are three examples of distinct programming interfaces currently provided by listeners.

Transport adapters insulate dispatchers from the proprietary protocols and interfaces used by web listeners. Specifically, each transport adapter is configured to recognize the protocols of different listeners, and to convert the browser requests received from the listeners into converted browser requests having a standard dispatcher protocol that is independent from the protocol of the listener. Similarly, transport adapters convert the replies from the dispatcher to the transport protocol of the listeners.

Hence, the transport adapter enables the web application server **280** to be used with listeners from different vendors. Moreover, transport adapters may be configured to accommodate different server architectures and operating systems.

OPERATION OF THE WEB APPLICATION SERVER

FIGS. **3A** and **3B** are a flow diagram illustrating a method of responding to a browser request according to an embodiment of the present invention. The browser request is received in step **350** by a listener. For the purposes of explanation, it shall be assumed that the browser request was issued by browser **202** and received by listener **210**.

Upon receiving the browser request, the listener **210** forwards the request to the web application server **280** in step **352**. Specifically, listener **210** passes the request to the transport adapter **212** using the proprietary programming interface of the listener **210**. The transport adapter **212** converts the request as necessary to pass the request to dispatcher **214** using a standard dispatcher programming interface.

Dispatcher **214** identifies the request object type that corresponds to the browser request in step **354** based on the virtual path specified in the browser request by communicating with the virtual path manager **250**. If the request object type corresponds to a cartridge, the virtual path manager also indicates to the dispatcher **214** whether authentication is required.

The dispatcher **214** determines in step **356** if the request object type corresponds to an identifiable cartridge. If the request object type does not correspond to an identifiable cartridge, the request is returned to the listener **210** in step **358** (see FIG. **3B**). If in step **358** the listener **210** recognizes the request as a request for a static HTML page, the listener accesses the static HTML page, and sends the HTML page to the browser **202** in step **360**. If the browser request is not recognized by the listener **210**, the reply is sent to the browser **202** in step **360** indicating that the request was unrecognizable.

If in step **356** the dispatcher **214** determines that the request must be sent to a cartridge, then the dispatcher performs any necessary authentication by communicating with the authentication server **252**. The authentication process will be described in greater detail hereafter. In addition, if in step **356** it is determined that listener **210** has not been assigned any instances of that cartridge that are currently FREE, then the dispatcher **214** communicates with the resource manager **254** to be assigned an instance of the cartridge **230** to which the browser request can be sent.

In step **362**, shown in FIG. **3B**, the resource manager **254** determines whether an instance of the identified cartridge is available (unowned) among the existing number of instances. For the purposes of explanation, it shall be assumed that the request is associated with cartridge **230**, and that cartridge **230** is a PL/SQL runtime cartridge.

If in step **362** the resource manager identifies an available instance, for example instance **260** of the PL/SQL runtime **230**, the resource manager **254** informs the dispatcher **214** that the request should be sent to instance **260**. The dispatcher **214** then creates and sends a revised browser request to the cartridge execution engine **228** of the instance **260** in step **368** to cause the available instance **260** to process the request, as described below.

However, if in step **362** no instance of the cartridge **230** is available, the resource manager **254** determines in step **364** if the existing number of instances exceeds a maximum prescribed number. If the existing number of instances exceeds the maximum prescribed number in step **364**, the resource manager **254** indicates to the dispatcher **214** that the request cannot be processed at this time. In response, the dispatcher **214** returns the request to the listener **210** in step **358**, after which the web listener **210** sends a reply to the browser **202** over the network in step **360** indicating the request was not processed.

Alternatively, when a cartridge instance is not currently available to handle a request, listener **210** may place the request on a waiting list for that cartridge instance. When a cartridge instance becomes available, the revised browser request is removed from the waiting list and forwarded to the cartridge instance. If the revised browser request remains on the waiting list for more than a predetermined amount of time, listener **210** may remove the request from the waiting

ORCL01622488

40. The system of claim 14, wherein

the step of said cartridge execution engine intercepting browser messages includes the step of said cartridge execution engine intercepting browser messages that include a rollback transaction command; and

in response to said cartridge execution engine receiving a browser message that includes a rollback transaction command, said cartridge execution engine sending a transaction control message to said transaction manager to cause said transaction manager to roll back said transaction.

41. The method of claim 1, further comprising the steps of:

prior to intercepting browser messages directed to said cartridge, storing metadata that establishes a correlation between one or more transaction commands and one or more message items;

in response to intercepting browser messages directed to said cartridge, comparing the message items that are within each browser message with said metadata to determine whether a particular transaction command needs to be executed; and

if it is determined that a particular transaction command does need to be executed, including data within a transaction control message that will cause said transaction manager to perform the particular transaction command.

42. The computer readable medium, of claim 11, further comprising instructions for performing the steps of:

prior to intercepting browser messages directed to said cartridge, storing metadata that establishes a correlation between one or more transaction commands and one or message items;

in response to intercepting browser messages directed to said cartidge, comparing the message items that are within each browser message with said metadata to determine whether a particular transaction command needs to be executed; and

if it is determined that a particular transaction command does need to be executed, including data within a transaction control message that will cause said transaction manager to perform the particular transaction command.

43. The system of claim 14, further comprising the steps of:

prior to intercepting browser messages directed to said cartridge, storing metadata that establishes a correlation between one or more transaction commands and one or more message items;

in response to intercepting browser messages directed to said cartridge, comparing the message items that are within each browser message with said metadata to determine whether a particular transaction command needs to be executed; and

if it is determined that a particular transaction command does need to be executed, including data within a transaction control message that will cause said transaction manager to perform the particular transaction command.

44. A method for executing a transaction that involves a series of operations, the method comprising the steps of:

registering said transaction by storing metadata that establishes a mapping between transactions commands and message items;

receiving a series of browser messages that request performance of said series of operations;

in response to said series of browser messages, executing said series of operations as an atomic unit of work; and

determining when to begin, commit and roll back said atomic unit of work based on message items in said series of browser messages and said metadata.

45. The method of claim 44, wherein the step of registering said transaction further comprises the step of storing metadata that establishes a belong-to-list, wherein the belong-to-list identifies a set of one or more cartridges that may participate in performing said transaction.

46. The method of claim 44, wherein the step of registering said transaction further comprises the step of storing metadata that establishes a resource-list, wherein the resource-list identifies a set of one or more resources that are affected by performing the transaction.

47. The method of claim 44, wherein the step of registering said transaction further comprises the step of storing metadata that establishes a cartridge name, wherein the cartridge name identifies a particular type of cartridges that may be used to perform the transaction.

48. The method of claim 44, wherein the step of registering said transaction further comprises the step of storing metadata that establishes a transaction name, wherein the transaction name uniquely identifies a type of transaction relative to other transaction types.

* * * * *

ORCL01622500

A8

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
90/008,562	03/29/2007	5894554	06-456	4189

8791 7590 05/04/2007

BLAKELY SOKOLOFF TAYLOR & ZAFMAN
12400 WILSHIRE BOULEVARD
SEVENTH FLOOR
LOS ANGELES, CA 90025-1030

EXAMINER	
Scott L. Weaver	
ART UNIT	**PAPER NUMBER**
3992	IFW

DATE MAILED: 05/04/2007

Please find below and/or attached an Office communication concerning this application or proceeding.

PTO-90C (Rev. 10/03)

ORCL00939453

UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

DO NOT USE IN PALM PRINTER

(THIRD PARTY REQUESTER'S CORRESPONDENCE ADDRESS)

MCDONNELL BOEHNEN HULBERT & BERGHOFF LLP

300 S. WACKER DRIVE

32ND FLOOR

CHICAGO, IL 60606

EX PARTE REEXAMINATION COMMUNICATION TRANSMITTAL FORM

REEXAMINATION CONTROL NO. *90/008,562*.

PATENT NO. *5894554*.

ART UNIT *3992*.

Enclosed is a copy of the latest communication from the United States Patent and Trademark Office in the above identified *ex parte* reexamination proceeding (37 CFR 1.550(f)).

Where this copy is supplied after the reply by requester, 37 CFR 1.535, or the time for filing a reply has passed, no submission on behalf of the *ex parte* reexamination requester will be acknowledged or considered (37 CFR 1.550(g)).

PTOL-465 (Rev.07-04)

Order Granting / Denying Request For Ex Parte Reexamination	Control No.	Patent Under Reexamination	
	90/008,562	5894554	
	Examiner	Art Unit	
	Scott L. Weaver	3992	

--The MAILING DATE of this communication appears on the cover sheet with the correspondence address--

The request for *ex parte* reexamination filed *29 March 2007* has been considered and a determination has been made. An identification of the claims, the references relied upon, and the rationale supporting the determination are attached.

Attachments: a)☐ PTO-892, b)☒ PTO/SB/08, c)☒ Other: *Decision on Request*

1. ☒ The request for *ex parte* reexamination is GRANTED.

RESPONSE TIMES ARE SET AS FOLLOWS:

For Patent Owner's Statement (Optional): TWO MONTHS from the mailing date of this communication (37 CFR 1.530 (b)). **EXTENSIONS OF TIME ARE GOVERNED BY 37 CFR 1.550(c).**

For Requester's Reply (optional): TWO MONTHS from the **date of service** of any timely filed Patent Owner's Statement (37 CFR 1.535). **NO EXTENSION OF THIS TIME PERIOD IS PERMITTED.** If Patent Owner does not file a timely statement under 37 CFR 1.530(b), then no reply by requester is permitted.

2. ☐ The request for *ex parte* reexamination is DENIED.

This decision is not appealable (35 U.S.C. 303(c)). Requester may seek review by petition to the Commissioner under 37 CFR 1.181 within ONE MONTH from the mailing date of this communication (37 CFR 1.515(c)). **EXTENSION OF TIME TO FILE SUCH A PETITION UNDER 37 CFR 1.181 ARE AVAILABLE ONLY BY PETITION TO SUSPEND OR WAIVE THE REGULATIONS UNDER 37 CFR 1.183.**

In due course, a refund under 37 CFR 1.26 (c) will be made to requester:

a) ☐ by Treasury check or,

b) ☐ by credit to Deposit Account No. _____, or

c) ☐ by credit to a credit card account, unless otherwise notified (35 U.S.C. 303(c)).

cc:Requester (if third party requester)

Decision on Request for Ex Parte Reexamination

Reexamination has been requested for claims 1-11 of United States Patent Number 5,894,554 to Lowery et al. issued on April 13, 1999 from application No. 08/636,477 filed on April 23, 1996.

A substantial new question of patentability affecting claims 1-11 of United States Patent Number 5,894,554 to Lowery is raised by the request for reexamination filed on March 29, 2007 for the reasons set forth below.

The References Cited in The Request

The Request identifies the following documents as providing teachings relevant to claims 1-11 of United States Patent Number 5,894,554 to Lowery

Exhibit E: "The World-Wide Web Gateway to Hyper-G: Using a Connectionless Protocol to Access Session-Oriented Services", Diploma Thesis of Christian Derler, Technology University of Graz, Austria, July 1, 1995 (hereafter "Derler")

Exhibit F: "Dienst: Implementation Reference Manual", Technical Report TR 95-1514, Lagoze et al., , Department of Computer Science, Cornell University, May 5, 1995 (hereafter "Dienst).

Each of the Derler and Dienst references has not previously been made of record during prosecution of the application which became the 5,894,554 patent to Lowery and as such has not been previously considered nor addressed during an 'examination' of the application which became the 5,894,554 patent to Lowery, nor in a final holding of invalidity by the Federal Courts. The Derler and Dienst references listed above are not cumulative to the prior art of record.

Issues Raised in the Request

The request indicates that the requestor considers:

> Claims 1-11 of the 5,894,554 patent to Lowery may be unpatentable over Derler alone with details provided on pages 8-21 of the request.
>
> Claims 1-5 and 7-11 of the 5,894,554 patent to Lowery may be unpatentable over Dienst alone with details provided on pages 22-35 of the request.
>
> Claim 6 of the 5,894,554 patent to Lowery may be unpatentable over Derler in combination with Dienst with details provided on page 29 of the request.

The 5,894,554 patent to Lowery discloses management of dynamic web page generation requests to a web server with the request intercepted and routed from web server to a page server such as to release the web server from processing the request and so that the web server may process other requests concurrently. A dynamically generated web page with data dynamically retrieved from one or more data sources is generated from the intercepted request and sent back to the requesting client or stored on machine accessible to web server for later retrieval (col.2,ln.21-33; col.4,ln.54-62; col.5,ln.38-48; col.6,ln.21-32).

Claim 1 is representative:

> 1. A computer-implemented method for managing a dynamic Web page generation request to a Web server, said computer-implemented method comprising the steps of:
> routing said request from said Web server to a page server, said page server receiving said request and releasing said Web server to process other requests, wherein said routing step further includes the steps of intercepting said request at said Web server, routing said request from said Web server to a dispatcher, and dispatching said request to said page server;
> processing said request, said processing being performed by said page server while said Web server concurrently processes said other requests; and
> dynamically generating a Web page in response to said request, said Web page including data dynamically retrieved from one or more data sources.

The examiner did not indicate reasons for allowance during prosecution of the application which became the 5,894,554 patent to Lowery. The record indicates that the prior art of record did not teach or suggest 'dynamically generating a Web page in response to a request wherein the Web page includes data dynamically retrieved from one or more data sources, as claimed'.

-Requester takes the position that Derler discloses a WWW (world wide web) Gateway which includes software features including Master, Slave, and Child processes which intercept and dispatch requests to the page server (a Hyper-G server) while the WWW Gateway processes other requests, the description on page 51, last paragraph through page 55 first paragraph and figures 4.3, 4.4., 4.5 of Derler describes the generation of dynamic web pages using architecture as shown in figure 4.17 on page 95.

It is agreed that the description of the system in Derler as presented in the request raise a substantial new question of patentability with respect to claim 1 of the 5,894,554 patent to Lowery . The Derler reference describing the dynamic generation of a web page using the Master, Slave, Child processes of the gateway and Hyper-G server as described on pages 11-14 was not before the office during any previous examination of the application which became the 5,894,554 patent to Lowery. There is a substantial likelihood that a reasonable examiner would have considered the Derler reference describing these features important in making a decision as to the patentability of claims 1-11 during the examination of the application which became the 5,894,554 patent to Lowery.

-Requester takes the position that Dienst discloses a system and method for managing requests for documents ("Web Pages") stored on page server (Dienst server) or other server from WWW clients. The requester indicates that Dienst discloses the requests from the client being intercepted at the server by a CGI (Common Gateway Interface) stub as shown via figure 2 on page 10 of Dienst. The CGI stub dispatches the requests to a page server (Dienst server) which retrieve the requested dynamic content for delivery to the requesting client and which allows the WWW server to handle other incoming requests.

It is agreed that the description of the system in Dienst as presented in the request raise a substantial new question of patentability with respect to claim 1 of the 5,894,554 patent to Lowery . The Dienst reference describing the dynamic generation of a web page using the Dienst Server , WWW server and CGI stub as described on pages 22-35 of the request was not before the office during any previous examination of the application which became the 5,894,554 patent to Lowery. There is a substantial likelihood that a reasonable examiner would have considered the Dienst reference describing the disclosed features noted above important in making a decision as to the patentability of claims 1-11 during the examination of the application which became the 5,894,554 patent to Lowery.

-Requester takes the position that Derler in combination with Dienst as discussed on page 29 of the request raises a substantial new question of patentability with respect to claims 6. Since each of Derler and Dienst alone is considered to raise a substantial new question of patentability as noted above, the combination proposed also raises a substantial new question of patentability for similar reasons.

All claims 1-11 of the 5,894,554 patent to Lowery will be reexamined as requested in the request filed on 03/29/2007.

Extensions of time under 37 CFR 1.136(a) will not be permitted in these proceedings because the provisions of 37 CFR 1.136 apply only to "an applicant" and not to parties in a reexamination proceeding. Additionally, 35 U.S.C. 305 requires that *ex parte* reexamination proceedings "will be conducted with special dispatch" (37 CFR 1.550(a)). Extensions of time in *ex parte* reexamination proceedings are provided for in 37 CFR 1.550(c).

The patent owner is reminded of the continuing responsibility under 37 CFR 1.565(a) to apprise the Office of any litigation activity, or other prior or concurrent proceeding, involving U.S. Patent Number 5,894,554 throughout the course of this reexamination proceeding. The third party requester is also reminded of the ability to similarly apprise the Office of any such activity or proceeding throughout the course of this reexamination proceeding. See MPEP §§ 2207, 2282 and 2286.

Please mail any communications to:
Attn: Mail Stop "Ex Parte Reexam"
Central Reexamination Unit
Commissioner for Patents
P. O. Box 1450
Alexandria VA 22313-1450

Please FAX any communications to:
(571) 273-9900
Central Reexamination Unit

Please hand-deliver any communications to:
Customer Service Window
Attn: Central Reexamination Unit
Randolph Building, Lobby Level
401 Dulany Street
Alexandria, VA 22314

Any inquiry concerning this communication or earlier communications from the Reexamination Legal Advisor or Examiner, or as to the status of this proceeding, should be directed to the Central Reexamination Unit at telephone number (571) 272-7705.

Scott L. Weaver
Primary Examiner
Central Reexam Unit 3992
(571) 272-7548

SCOTT L. WEAVER
CRU EXAMINER-AU 3992

Conferees:

ROLAND G. FOSTER
CRU EXAMINER-AU 3992

MARK J. REINHART
SPRE-AU 3992
CENTRAL REEXAMINATION UNIT

NOTICE RE PATENT OWNER'S CORRESPONDENCE ADDRESS

Effective May 16, 2007, 37 CFR 1.33(c) has been revised to provide that:

The patent owner's correspondence address for all communications in an *ex parte* reexamination or an *inter partes* reexamination is designated as the correspondence address of the patent.

> *Revisions and Technical Corrections Affecting Requirements for Ex Parte and Inter Partes Reexamination*, 72 FR 18892 (April 16, 2007)(Final Rule)

The correspondence address for any pending reexamination proceeding not having the same correspondence address as that of the patent is, by way of this revision to 37 CFR 1.33(c), automatically changed to that of the patent file as of the effective date.

This change is effective for any reexamination proceeding which is pending before the Office as of May 16, 2007, including the present reexamination proceeding, and to any reexamination proceeding which is filed after that date.

Parties are to take this change into account when filing papers, and direct communications accordingly.

In the event the patent owner's correspondence address listed in the papers (record) for the present proceeding is different from the correspondence address of the patent, it is strongly encouraged that the patent owner affirmatively file a Notification of Change of Correspondence Address in the reexamination proceeding and/or the patent (depending on which address patent owner desires), to conform the address of the proceeding with that of the patent and to clarify the record as to which address should be used for correspondence.

Telephone Numbers for reexamination inquiries:

Reexamination and Amendment Practice	(571) 272-7703
Central Reexam Unit (CRU)	(571) 272-7705
Reexamination Facsimile Transmission No.	(571) 273-9900

C:\kiva\kenreex\ RRR-new-PO-address_by-rule.doc
May 1, 2007

ORCL00939460

A9

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
90/008,568	03/29/2007	6415335	06-455	7319

mj

7590 05/04/2007

MATTHEW B. TALPIS, ESQ.
BAKER BOTTS LLP
2001 ROSS AVENUE
SUITE 600
DALLAS, TX 75201

EXAMINER	
Scott L. Weaver	
ART UNIT	**PAPER NUMBER**
3992	IFW

DATE MAILED: 05/04/2007

Please find below and/or attached an Office communication concerning this application or proceeding.

PTO-90C (Rev. 10/03)

UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

DO NOT USE IN PALM PRINTER

(THIRD PARTY REQUESTER'S CORRESPONDENCE ADDRESS)

MCDONNELL BOEHNEN HULBERT & BERGHOFF LLP

300 S. WACKER DRIVE

32ND FLOOR

CHICAGO, IL 60606

EX PARTE REEXAMINATION COMMUNICATION TRANSMITTAL FORM

REEXAMINATION CONTROL NO. *90/008,568*.

PATENT NO. *6415335*.

ART UNIT *3992*.

Enclosed is a copy of the latest communication from the United States Patent and Trademark Office in the above identified *ex parte* reexamination proceeding (37 CFR 1.550(f)).

Where this copy is supplied after the reply by requester, 37 CFR 1.535, or the time for filing a reply has passed, no submission on behalf of the *ex parte* reexamination requester will be acknowledged or considered (37 CFR 1.550(g)).

PTOL-465 (Rev.07-04)

Order Granting / Denying Request For Ex Parte Reexamination	Control No.	Patent Under Reexamination	
	90/008,568	6415335	
	Examiner	Art Unit	
	Scott L. Weaver	3992	

--The MAILING DATE of this communication appears on the cover sheet with the correspondence address--

The request for *ex parte* reexamination filed *29 March 2007* has been considered and a determination has been made. An identification of the claims, the references relied upon, and the rationale supporting the determination are attached.

Attachments: a)☐ PTO-892, b)☒ PTO/SB/08, c)☒ Other: *Decision on Request*

1. ☒ The request for *ex parte* reexamination is GRANTED.

RESPONSE TIMES ARE SET AS FOLLOWS:

For Patent Owner's Statement (Optional): TWO MONTHS from the mailing date of this communication (37 CFR 1.530 (b)). **EXTENSIONS OF TIME ARE GOVERNED BY 37 CFR 1.550(c).**

For Requester's Reply (optional): TWO MONTHS from the **date of service** of any timely filed Patent Owner's Statement (37 CFR 1.535). **NO EXTENSION OF THIS TIME PERIOD IS PERMITTED.** If Patent Owner does not file a timely statement under 37 CFR 1.530(b), then no reply by requester is permitted.

2. ☐ The request for *ex parte* reexamination is DENIED.

This decision is not appealable (35 U.S.C. 303(c)). Requester may seek review by petition to the Commissioner under 37 CFR 1.181 within ONE MONTH from the mailing date of this communication (37 CFR 1.515(c)). **EXTENSION OF TIME TO FILE SUCH A PETITION UNDER 37 CFR 1.181 ARE AVAILABLE ONLY BY PETITION TO SUSPEND OR WAIVE THE REGULATIONS UNDER 37 CFR 1.183.**

In due course, a refund under 37 CFR 1.26 (c) will be made to requester:

a) ☐ by Treasury check or,

b) ☐ by credit to Deposit Account No. _____, or

c) ☐ by credit to a credit card account, unless otherwise notified (35 U.S.C. 303(c)).

cc:Requester (if third party requester)

Decision on Request for Ex Parte Reexamination

Reexamination has been requested for claims 1-29 of United States Patent Number 6,415,335 to Lowery et al. issued on July 2, 2002 from divisional application No. 09/234,048 claiming priority to the 5,894,554 patent to Lowery filed on April 23, 1996 and subject of reexamination control number 90/008,342.

A substantial new question of patentability affecting claims 1-29 of United States Patent Number 6,415,3356 to Lowery is raised by the request for reexamination filed on March 29, 2007 for the reasons set forth below.

The References Cited in The Request

The Request identifies the following documents as providing teachings relevant to claims 1-29 of United States Patent Number 6,415,335 to Lowery.

Exhibit E: "The World-Wide Web Gateway to Hyper-G: Using a Connectionless Protocol to Access Session-Oriented Services", Diploma Thesis of Christian Derler, Technology University of Graz, Austria, July 1, 1995 (hereafter "Derler")

Exhibit F: "Dienst: Implementation Reference Manual", Technical Report TR 95-1514, Lagoze et al., , Department of Computer Science, Cornell University, May 5, 1995 (hereafter "Dienst).

Exhibit G: U.S. Patent 5,682,468 to Fortenbery et al.

Each of the references cited as the exhibits E, F, and G above has not previously been made of record during prosecution of the application which became the 6,415,335 patent to Lowery and as such has not been previously considered nor addressed during an 'examination' of the application which became the 6,415,335 patent to Lowery, nor in a final holding of invalidity by the Federal Courts. The references listed above are not cumulative to the prior art of record.

Issues Raised in the Request

The request indicates that the requestor considers:

-Claims 1-12, 15-26 and 29 of the 6,415,3356 patent to Lowery may be unpatentable over Derler alone with details provided on pages 9-26 of the request.

-Claims 13-14, and 27-28 of the 6,415,3356 patent to Lowery may be unpatentable over Derler in combination with Fortenbery with details provided on pages 17-18, and 23-24 of the request.

-Claims 1-6, 8-12, 15-26 and 29 of the 6,415,3356 patent to Lowery may be unpatentable over Dienst alone with details provided on pages 27-43 of the request.

-Claims 7 and 21 of the 6,415,3356 patent to Lowery may be unpatentable over Dienst in combination with Derler with details provided on pages 33 and 39 of the request.

-Claims 13-14, and 27-28 of the 6,415,3356 patent to Lowery may be unpatentable over Dienst in combination with Fortenbery with details provided on pages 35-36 and 41-42 of the request.

The 6,415,3356 patent to Lowery discloses management of dynamic web page generation requests to a web server with the request intercepted and routed from web server to a page server such as to release the web server from processing the request and so that the web server may process other requests concurrently. A dynamically generated web page with data dynamically retrieved from one or more data sources is generated from the intercepted request and sent back to the requesting client or stored on machine accessible to web server for later retrieval (col. 2, ln.. 21-32; col. 4, ln. 54-62; col. 5, ln. 38-48; col. 6 ,ln. 19-31).

Claim 1 is representative:

1. A computer-implemented method for managing a dynamic Web page generation request to a Web server, said computer-implemented method comprising the steps of:
routing a request from a Web server to a page server, said page server receiving said request and releasing said Web server to process other requests wherein said routing step further includes the steps of:
intercepting said request at said Web server and routing said request to said page server;
processing said request, said processing being performed by said page server while said Web server concurrently processes said other requests;
and
dynamically generating a Web page in response to said request, said Web page

including data dynamically retrieved from one or more data sources.

2. The computer-implemented method in claim 1 wherein said step of routing said request includes the steps of:
routing said request from said Web server to a dispatcher; and
dispatching said request to said page server.

The examiner did not indicate a reasons for allowance during prosecution of the application which became the 6,415,335 patent to Lowery.

-Requester takes the position that Derler discloses a WWW (world wide web) Gateway which includes software features including Master, Slave, and Child processes which intercept and dispatch requests to the page server (a Hyper-G server) while the WWW Gateway processes other requests. On page 51, last paragraph through page 55 first paragraph and figures 4.3, 4.4., 4.5, Derler describes the generation of dynamic web pages using architecture as shown in figure 4.17 on page 95.

It is agreed that the description of the system in Derler as presented in the request raise a substantial new question of patentability with respect to claim 1 of the 6,415,335 patent to Lowery . The Derler reference describing the dynamic generation of a web page using the Master, Slave, Child processes of the gateway and Hyper-G server as described on pages 9-11 was not before the office during any previous examination of the application which became the 6,415,335 patent to Lowery. There is a substantial likelihood that a reasonable examiner would have considered the Derler reference describing these features important in making a decision as to the patentability of claims 1-29 during the examination of the application which became the 6,415,335 patent to Lowery.

-Requester takes the position that Dienst discloses a system and method for managing requests for documents ("Web Pages") stored on page server (Dienst server) or other server from WWW clients. The requester indicates that Dienst discloses the requests from the client being intercepted at the server by a CGI (Common Gateway Interface) stub as shown via figure 2 on page 10 of Dienst. The CGI stub dispatches the requests to a page server (Dienst server) which retrieve the requested dynamic content for delivery to the requesting client and which allows the WWW server to handle other incoming requests.

It is agreed that the description of the system in Dienst as presented in the request raise a substantial new question of patentability with respect to claim 1 of the 6,415,335 patent to Lowery . The Dienst reference describing the dynamic generation of a web page using the Dienst Server , WWW server and CGI stub as described on pages 27-43 of the request was not before

the office during any previous examination of the application which became the 6,415,335 patent to Lowery. There is a substantial likelihood that a reasonable examiner would have considered the Dienst reference describing the noted features important in making a decision as to the patentability of claims 1-29 during the examination of the application which became the 6,415,335 patent to Lowery.

-Requester takes the position that Derler in combination with Dienst as discussed on pages 33 and 39 of the request and that Derler or Dienst separately in combination with Fortenbery raise a substantial new question of patentability with respect to claims noted above. Since each of Derler and Dienst alone is considered to raise a substantial new question of patentability as noted above, the combinations proposed also raises a substantial new question of patentability for similar reasons.

All claims 1-29 of the 6,415,335 patent to Lowery will be reexamined as requested in the request filed on 3/29/2007.

Extensions of time under 37 CFR 1.136(a) will not be permitted in these proceedings because the provisions of 37 CFR 1.136 apply only to "an applicant" and not to parties in a reexamination proceeding. Additionally, 35 U.S.C. 305 requires that *ex parte* reexamination proceedings "will be conducted with special dispatch" (37 CFR 1.550(a)). Extensions of time in *ex parte* reexamination proceedings are provided for in 37 CFR 1.550(c).

The patent owner is reminded of the continuing responsibility under 37 CFR 1.565(a) to apprise the Office of any litigation activity, or other prior or concurrent proceeding, involving U.S. Patent Number 6,415,335 throughout the course of this reexamination proceeding. The third party requester is also reminded of the ability to similarly apprise the Office of any such activity or proceeding throughout the course of this reexamination proceeding. See MPEP §§ 2207, 2282 and 2286.

Please mail any communications to:
Attn: Mail Stop "Ex Parte Reexam"
Central Reexamination Unit
Commissioner for Patents
P. O. Box 1450
Alexandria VA 22313-1450

Please FAX any communications to:
(571) 273-9900
Central Reexamination Unit

Please hand-deliver any communications to:
Customer Service Window
Attn: Central Reexamination Unit
Randolph Building, Lobby Level
401 Dulany Street
Alexandria, VA 22314

Any inquiry concerning this communication or earlier communications from the Reexamination Legal Advisor or Examiner, or as to the status of this proceeding, should be directed to the Central Reexamination Unit at telephone number (571) 272-7705.

Scott L. Weaver
Primary Examiner
Central Reexam Unit 3992
(571) 272-7548

SCOTT L. WEAVER
CRU EXAMINER-AU 3992

Conferees:

ROLAND G. FOSTER
CRU EXAMINER-AU 3992

MARK J. REINHART
SPRE-AU 3992
CENTRAL REEXAMINATION UNIT

NOTICE RE PATENT OWNER'S CORRESPONDENCE ADDRESS

Effective May 16, 2007, 37 CFR 1.33(c) has been revised to provide that:

The patent owner's correspondence address for all communications in an *ex parte* reexamination or an *inter partes* reexamination is designated as the correspondence address of the patent.

> *Revisions and Technical Corrections Affecting Requirements for Ex Parte and Inter Partes Reexamination*, 72 FR 18892 (April 16, 2007)(Final Rule)

The correspondence address for any pending reexamination proceeding not having the same correspondence address as that of the patent is, by way of this revision to 37 CFR 1.33(c), automatically changed to that of the patent file as of the effective date.

This change is effective for any reexamination proceeding which is pending before the Office as of May 16, 2007, including the present reexamination proceeding, and to any reexamination proceeding which is filed after that date.

Parties are to take this change into account when filing papers, and direct communications accordingly.

In the event the patent owner's correspondence address listed in the papers (record) for the present proceeding is different from the correspondence address of the patent, it is strongly encouraged that the patent owner affirmatively file a Notification of Change of Correspondence Address in the reexamination proceeding and/or the patent (depending on which address patent owner desires), to conform the address of the proceeding with that of the patent and to clarify the record as to which address should be used for correspondence.

Telephone Numbers for reexamination inquiries:

Reexamination and Amendment Practice	(571) 272-7703
Central Reexam Unit (CRU)	(571) 272-7705
Reexamination Facsimile Transmission No.	(571) 273-9900

C:\kiva\kenreex\ RRR-new-PO-address_by-rule.doc
May 1, 2007

A10

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
90/008,574	04/03/2007	5894554	FRIENDFINDER.RX1	2817

8791 7590 06/30/2008

BLAKELY SOKOLOFF TAYLOR & ZAFMAN LLP
1279 OAKMEAD PARKWAY
SUNNYVALE, CA 94085-4040

EXAMINER	
ART UNIT	PAPER NUMBER

DATE MAILED: 06/30/2008

Please find below and/or attached an Office communication concerning this application or proceeding.

UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

DO NOT USE IN PALM PRINTER

(THIRD PARTY REQUESTER'S CORRESPONDENCE ADDRESS)

INTELLECTUAL PROPERTY LAW GROUP LLP
12 SOUTH FIRST STREET
SUITE 1205
SAN JOSE, CA 95113

MAILED

JUN 30 2008

CENTRAL REEXAMINATION UNIT

EX PARTE REEXAMINATION COMMUNICATION TRANSMITTAL FORM

REEXAMINATION CONTROL NO. *90/008,574*.

PATENT NO. *5894554*.

ART UNIT *3992*.

Enclosed is a copy of the latest communication from the United States Patent and Trademark Office in the above identified *ex parte* reexamination proceeding (37 CFR 1.550(f)).

Where this copy is supplied after the reply by requester, 37 CFR 1.535, or the time for filing a reply has passed, no submission on behalf of the *ex parte* reexamination requester will be acknowledged or considered (37 CFR 1.550(g)).

Office Action in Ex Parte Reexamination	Control No. 90/008,574 (90 008562, 90 008342)	Patent Under Reexamination 5894554	
	Examiner MARY STEELMAN	Art Unit 3992	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

a☒ Responsive to the communication(s) filed on *13 May 2008* . b☐ This action is made FINAL.
c☒ A statement under 37 CFR 1.530 has not been received from the patent owner.

A shortened statutory period for response to this action is set to expire 2 month(s) from the mailing date of this letter. Failure to respond within the period for response will result in termination of the proceeding and issuance of an *ex parte* reexamination certificate in accordance with this action. 37 CFR 1.550(d). **EXTENSIONS OF TIME ARE GOVERNED BY 37 CFR 1.550(c).** If the period for response specified above is less than thirty (30) days, a response within the statutory minimum of thirty (30) days will be considered timely.

Part I THE FOLLOWING ATTACHMENT(S) ARE PART OF THIS ACTION:

1. ☐ Notice of References Cited by Examiner, PTO-892.
2. ☒ Information Disclosure Statement, PTO/SB/08.
3. ☐ Interview Summary, PTO-474.
4. ☐ _____.

Part II SUMMARY OF ACTION

1a. ☒ Claims *1-11* are subject to reexamination.
1b. ☐ Claims _____ are not subject to reexamination.
2. ☐ Claims _____ have been canceled in the present reexamination proceeding.
3. ☐ Claims _____ are patentable and/or confirmed.
4. ☒ Claims *1-11* are rejected.
5. ☐ Claims _____ are objected to.
6. ☐ The drawings, filed on _____ are acceptable.
7. ☐ The proposed drawing correction, filed on _____ has been (7a)☐ approved (7b)☐ disapproved.
8. ☐ Acknowledgment is made of the priority claim under 35 U.S.C. § 119(a)-(d) or (f).
   a)☐ All b)☐ Some* c)☐ None of the certified copies have
   1☐ been received.
   2☐ not been received.
   3☐ been filed in Application No. _____ .
   4☐ been filed in reexamination Control No. _____.
   5☐ been received by the International Bureau in PCT application No. _____.
   * See the attached detailed Office action for a list of the certified copies not received.
9. ☐ Since the proceeding appears to be in condition for issuance of an *ex parte* reexamination certificate except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte* Quayle, 1935 C.D. 11, 453 O.G. 213.
10. ☐ Other: _____

cc: Requester (if third party requester)

DETAILED ACTION

I. Summary

This Office Action addresses the claims for which reexamination has been requested and a substantial new question of patentability has been determined to exist, specifically, claims 1-11 of U.S. 5,894,554 issued to Lowery et al.

II. Notice Regarding Certain Reexamination Issues

The patent owner is reminded of the continuing responsibility under 37 CFR 1.565(a), to apprise the Office of any litigation activity, or other prior or concurrent proceeding, involving Patent No. 5,894,554 throughout the course of this reexamination proceeding. See MPEP §§ 2207, 2282 and 2286.

Extensions of time under 37 CFR 1.136(a) will not be permitted in these proceedings because the provisions of 37 CFR 1.136 apply only to "a Patent Applicant" and not to parties in a reexamination proceeding. Additionally, 35 U.S.C. 305 requires that reexamination proceedings "will be conducted with special dispatch" (37 CFR 1.550(a)). Extension of time in *ex parte* reexamination proceedings are provided for in 37 CFR 1.550(c).

Information Disclosure Statement

Where patents, publications, and other such items of information are submitted by a party (patent owner or requester) in compliance with the requirements of the rules, the requisite degree of consideration to be given to such information will be normally limited by the degree to which the party filing the information citation has explained the content and relevance of the information. The initials of the examiner placed adjacent to the citations on the form PTO /SB /08A and 08B or its equivalent, without an indication to the contrary in the record, do not signify that the information has been considered by the examiner any further than to the extent noted above.

Cited references lacking dates or not found in the file have been lined through.

III. References Cited

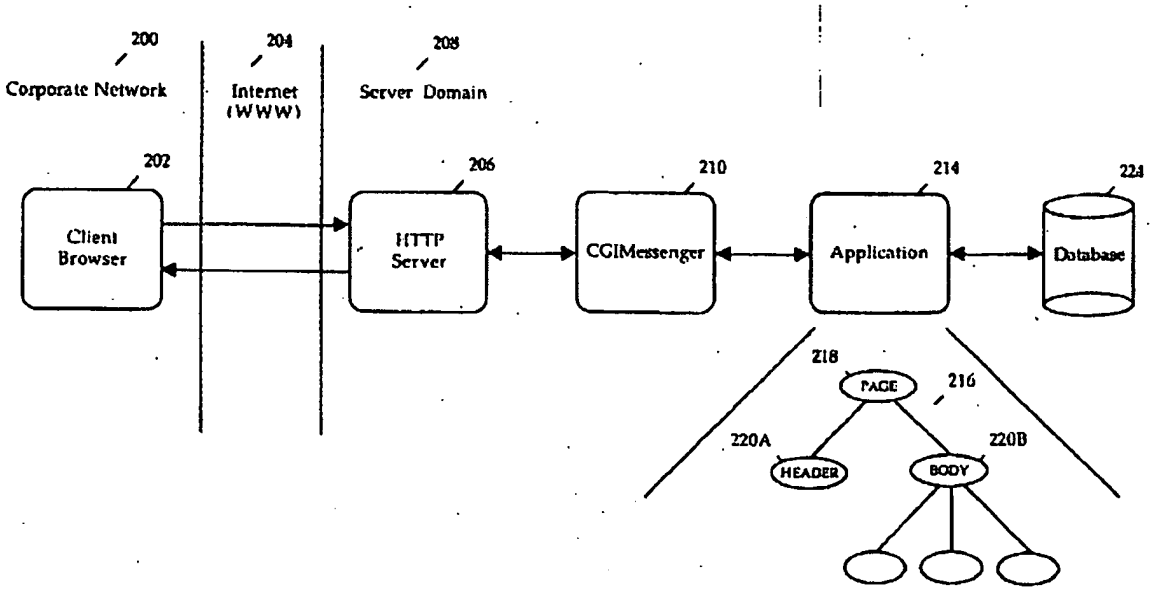
Rogers et al., USPN 5,701,451, file date Jun 7, 1995, issue date Dec 23, 1997

> A World Wide Web browser makes requests to web servers on a network which receive and fulfill requests as an agent of the browser client, organizing distributed sub-agents sub-agents as distributed integration solution (DIS) servers on an Intranet network supporting the web server which also has access agent servers accessible over the Internet. DIS servers execute selected capsule objects which perform programmable functions upon a received command from a web server control program agent for retrieving, from a database gateway coupled to a plurality of database resources upon a single request made from a Hypertext document, requested

information from multiple data bases located at different types of databases geographically dispersed, performing calculations, formatting, and other services prior to reporting to the web browser or to other locations, in a selected format, as in a display, fax, printer, and to customer installations or to TV video subscribers, with account tracking. (Abstract)

Popp et al., USPN 6,249,291 B1, file date Sep 22, 1995, issue date Jun 19, 2001

Popp discloses a system and method that "provides the ability to develop and manage Internet transactions." (Col. 3, 11.34-36.) In particular, a Web client can issue a request for a Web page to a Web server and "[s]ome or all of [the] Web page can be generated dynamically using input ... retrieved from an external data source (e.g., database ...)." (See, Abstract.) The Popp architecture is illustrated in Figure 2 of Popp (reproduced below). It comprises an HTTP server 206 (i.e., Web server), a CGIMessenger 210 and one or more applications 214 (i.e., page servers). (See generally, col. 6:32 - col. 7: 35.) Again, this architecture is substantially the same as the architecture described and claimed in the 554 patent; compare Figure 2 of Popp below to Figure 4 of the 544 patent.



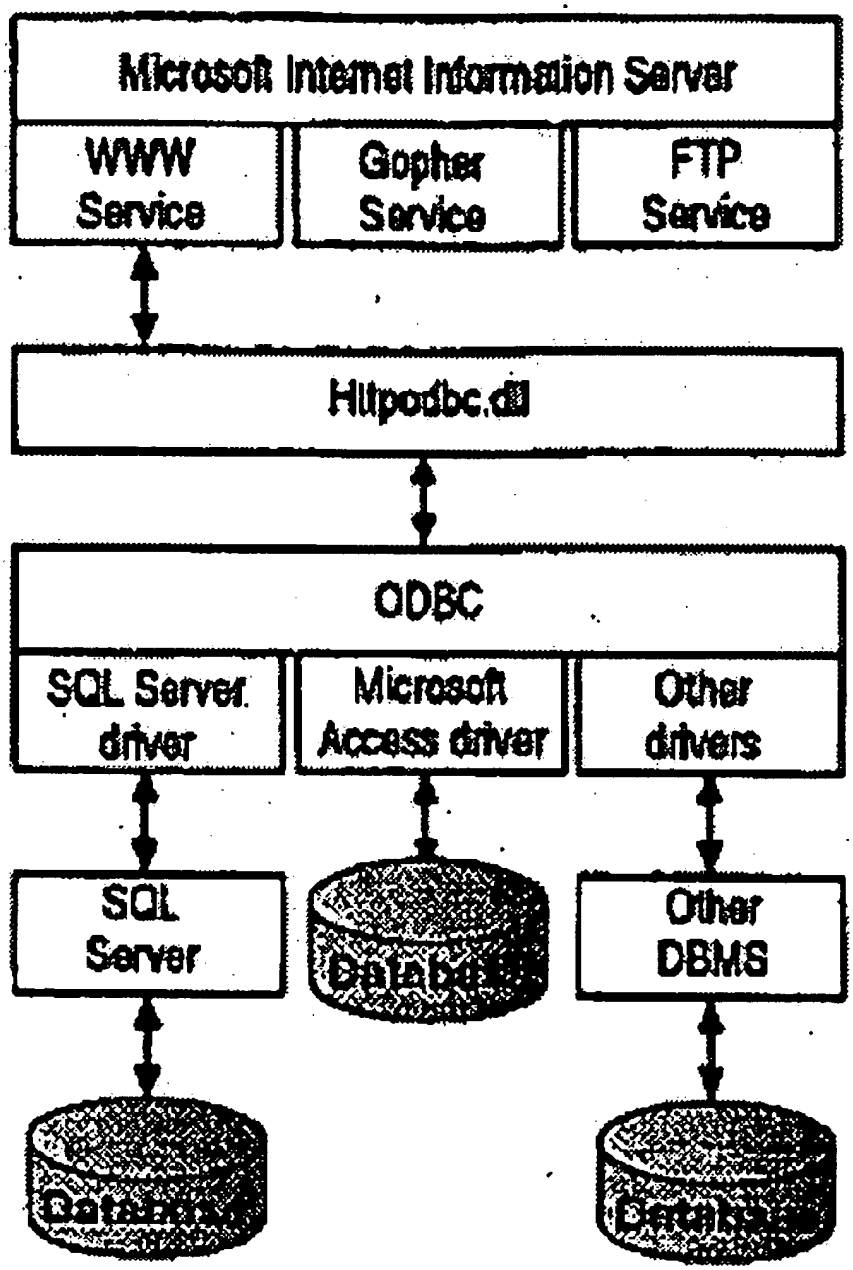
As described in Popp, a request from a Client Browser 202 (i.e., Web browser) is received by the Web server 206. If it is determined from the URL of the request that the request calls for the generation of dynamic content, the "HTTP server 206 initiates CGIMessenger 210." (Col. 7:13 - 22.) That is, processing of the request by the HTTP server 206 is stopped, deflected or interrupted and the CGIMessenger 210 takes over. If the request identifies an application, such as application 214, "the CGIMessenger 210 transmits the client request ... from HTTP Server 206 to application 214." (Col. 7:25-28.) Thus, the CGIMessenger 210 functions as a "dispatcher" to dispatch the request to the application identified in the URL of the request. (Col. 7:5-7) (the "common gateway interface program ... is used to route requests to applications."). Inherent in the CGIMessenger 210 intercepting the request and forwarding it to application 214 (i.e., a page server), the Web server 206 is released from processing that request and can process other requests. Indeed, "CGIMessenger 210 can execute on the HTTP server

206 or another server connected to HTTP server 206, for example." (Col. 6: 57-58) "Application 214 executes an interaction flow to satisfy the user request." Like the CGIMessenger 210, "[a]pplication 214 can execute on the same or different server as CGIMessenger 210 and/or HTTP Server 206." (Col. 7: 24-31.) The application 214 returns "a Web page (e.g., an HTML Web page) that is dynamically generated using complex queries (or other data retrieval mechanisms) to retrieve data and dynamically generate an HTML page using complex logic." (Col. 7: 44 - 58.) "Application 214 can access an external data source such as database 224. Database 224 can be resident on the same server as application 214 (Col. 7:31- 35.) In addition to the foregoing features, Popp also teaches that the system can be used to provide the notion of a "session" between a Web client and an application. Specifically, "[a] client's login password can be retained as state information at the session level, for example." (Col. 13: 33-35.) "Thereafter, the client does not have to enter the password to access an application during that session." Col. 13, 11.35-37. Thus, Popp teaches that the processing of requests may include the step of logging into (using a password, for example) the data source that an application accesses. Popp also teaches the use of HTML extension templates. The present invention provides an extension to HTML that is used on the HTTP server side. The HTML extension is filtered out before a Web page is sent to a client browser. It is used to interpret an HTML template and to render an HTML document before is transmitted to the client browser." (Col. 15: 45-50.) Specifically, Popp "uses an extension to the standard HTML known as the group extension. The group extension provides the ability to combine HTML elements or statements in a

single block." (Col. 15: 55-58.) "The group extension is identified by the <NSWTAG> and </NSWTAG> delimiters." (Col. 16: 31-32.) "[T]he group extension provides a link between an object that implements an HTML element and an object that implements a data item stored in a data source external to the WWW application." (Col. 15: 58 - 61.)

Installation and Planning Guide for Microsoft Internet Information Server, Version 1.0 ("Installation Guide") (published Aug 1995) supplemented with ODBC Web Database Add-on for Microsoft Internet Server Beta Release Notes ("ODBC Notes") (Sep 27, 1995) (Microsoft References)

Both of the Microsoft References describe features of Microsoft Internet Information Server (or simply "Microsoft Internet Server" or "Microsoft Web Server" or "MS IIS"). The Microsoft References disclose that "[w]ith the Web Server and ODBC, [one] can create Web pages with information contained in a database," thus enabling generation of dynamic Web pages. (ODBC Notes, p. 1.) As shown in the following figure from the Installation Guide, Microsoft Internet Information Server with ODBC Add-on consists of multiple components: the Microsoft Internet Information Server with WWW Service; Httpodbc.dll (also called Database Connector), ODBC with SQL Server driver; SQL Server; and multiple databases. This architecture is virtually identical to the architecture for offloading generation of' dynamic web pages disclosed and claimed in the '554 patent, Indeed, compare the figure below to Figure 4 of the 554 patent reproduced above.



As described in the Microsoft References, "Web browsers.., submit requests to the Internet server by using HyperText Transport Protocol (HTTP). The Internet Server responds: with a document formatted in HyperText Markup Language (HTML). Access to databases is accomplished through a component of the Internet Server called Internet

Database Connector, one of family of Internet Server Extensions. The Database Connector is a Dynamic-Link Library (DLL) file, Httpodbc.dll, that uses ODBC to access databases. Httpodbc.dll uses two types of files to control how the database is accessed and how the output Web page is constructed. These files are Web Database Gateway (WDG) files and HTML extension (HTX) files. The WDG files contain the necessary information to connect to the desired ODBC data source and execute the SQL statement. A WDG file also contains the name and location of the HTX file.

The HTX file is the template for the actual HTML document that will be returned to the Web Browser after the database information has been merged into it by Httpodbc.dll. (ODBC Notes, p. 2.) "On the Internet Server, the entire process of using the Database Connector is performed in six steps... Step 1" The URL is received by the Internet Server. The URL sent by the Web browser actually contains the name of the Database Connector, Httpodbc.dll, and the name of the WDG file to be used." (ODBC Notes, p. 4.) "Step 2: The Internet Server loads Httpodbc.dll and provides it with the remaining information in the URL... Step 3: Httpodbc.dll reads the WDG file... Step 4: Httpodbc.dll connects to the ODBC data source, and executes the SQL statement contained within the WDG file. The connection is made to the ODBC data source by Httpodbc.dll, which in this example loads the ODBC driver for SQL server and connects to the server specified in the definition of the data source. Once the connection is made, the SQL Statement in the WDG file is sent to the SQL Server ODBC driver, which in turn sends it to SQL Server. Step 5: Httpodbc.dll fetches the data from the database, and merges it into the HTX file. Step 6: Httpodbc.dll sends the merged document back to the Internet Server,

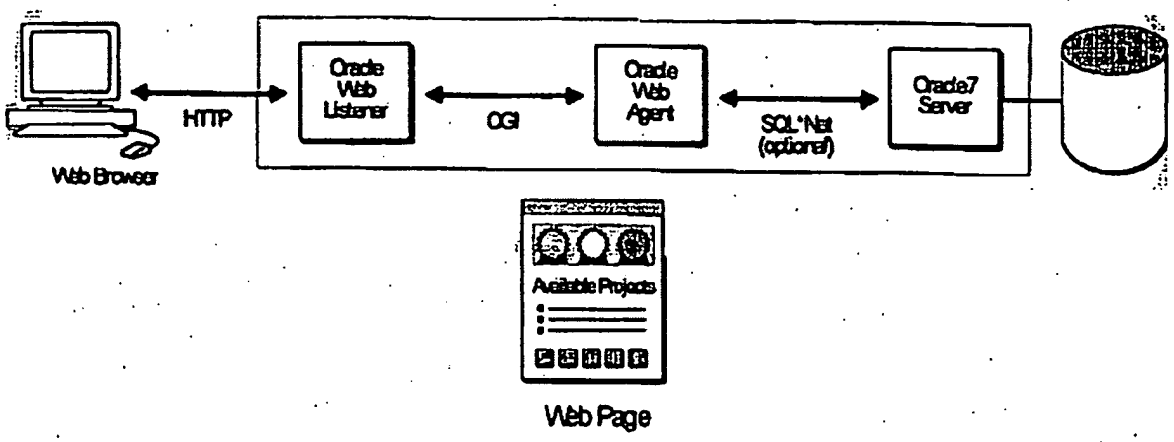
which returns it to the client." (ODBC Notes, pp. 3-8.) As applied to the claims of the '554 patent, Microsoft Internet Information Server with WWW Server is a Web Server. It examines an incoming request to determine whether the request was made for a static or a dynamic page, and if the request is made for a dynamic page, it further determines which DLL file is specified (such as Httpodbc.dll) and then interrupts its processing of the request. When "the Database Connector Httpodbc.dll is referenced in the URL," the Web Server will then act as a dispatcher to dispatch the request to the Database Connector Httpodbc.dll, which handles generation of dynamic Web pages. (ODBC Notes, p. 3) Installation Guide at pp. 21-22 discloses that Microsoft Internet Information Server is capable of supporting multiple simultaneous connections. Typically, a single process (or thread) monitors TCP port 80 (the default port for Web requests) for requests. When a requests comes on port 80, the process that monitors the port necessarily has to spawn a different process. Otherwise, other requests that arrive to the Web Server on port 80 would not be processed until that single process finishes processing the previous request, thus rendering simultaneous processing of multiple connections impossible. Accordingly, to support multiple simultaneous connections releasing must necessarily occur. In Step 2 quoted above, "The Internet Server loads Httpodbc.dll and provides it with the remaining information in the URL," thus accomplishing the step of routing said request from said Web server to a page server. In Step 3 quoted above, "The Connection is made to the ODBC data source by Httpodbc.dll, which in this example loads the ODBC driver for SQL Server and connects to the server specified in the definition of the data source. Once the connection is made, the SQL Statement in the WDG file is sent to the SQL Server

ODBC driver, which in turn sends it to SQL Server." SQL Server serves as a data source for the dynamically generated web page that Httpodbc.dll creates. On pp. 21-22, Installation Guide discloses that Microsoft Internet Information Server is capable of supporting simultaneous connections, thus it is capable of processing requests while ODBC driver for SQL server simultaneously processes other requests. In steps 4, 5, and 6 quoted above, Httpodbc.dll retrieves data from one or more data sources and dynamically generates a Web page that includes the retrieved data. "After the SQL statement has been executed, Httpodbc.dll reads the HTX file specified in Sample.wdg (Sample.htx) and replaces every placeholder with the actual data returned by the query." (ODBC Notes, pp. 7-8.) While processing the request according to the six steps described above, Microsoft Internet Information Server with ODBC Add-on identifies a data source from which data is retrieved. Additionally, the Microsoft References disclose logging into data sources because a username is provided in a query. Microsoft Internet Information Server with ODBC maintains a connection cache to data sources. Specifically, Microsoft Internet Information Server enables logging of events into data sources. "Choose Log to SQL/ODBC Database to log activity information to any O.DBC data source. Set the Datasource name, Table, and specify the username and password to use when logging to the database." (Installation Guide, p. 59.) Maintaining a connection cache is inherent because it is impracticable to open and close a connection to a data source for every log entry. Microsoft Internet Information Server with ODBC maintains a page cache: "By default, Httpodbc.dll caches the Web page sent to the client. If a subsequent request is identical, the cached page will be returned without ever accessing the database." (ODBC

Notes, p. 12.) The Microsoft References also disclose HTML extension templates for configuring Web pages. "The Sample.htx file is an HTML document that contains placeholders for data returned from the database." (ODBC Notes, p. 7.) Data retrieved from data sources is inserted into those templates. "After the SQL statement has been executed, Httpodbc.dll reads the HTX file specified in Sample.wdg (Sample.htx) and replaces every placeholder with the actual data returned by the query." (ODBC Notes, p. 7.)

Oracle® Web Server User's Guide, Release 1.0 (1995) and Katrina Montinola, "The Oracle WebServer: A Technical Discussion," September 25, 1995 (collectively "the Oracle references")

Both of the Oracle References describe Release 1.0 of Oracle's Web Server product. "The Oracle WebServer is an HTTP server with a tightly integrated Oracle 7 server that enables the creation of dynamic HTML documents from data stored in an Oracle database." (User's Guide, p. 1-2); see also, Montinola article, p. 2 ("the Oracle WebServer ... facilitates the creation of dynamic HTML documents ..."). As shown in the following figure from the User's Guide, the Oracle WebServer product consists of multiple components that closely correspond to the components of the partitioned architecture described and claimed in the 554 patent: the Oracle Web Listener, the Oracle Web Agent and the Oracle 7 Server. (User's Guide, p. 1-2; Montinola article, p. 1.) Indeed, a comparison of the figure below to Figure 4 of the 554 patent reproduced above reveals the similarities.



"The Oracle Web Listener is a commercial quality HTTP server that services document requests from any Web browser.... When the Oracle Web Listener receives a request from a client, it first determines whether that request is for a static document or a dynamic document." (User's Guide, p. 1-3; Montinola article, p. 3.) If the request is for a dynamic document, the Web Listener stops processing the request and invokes the Oracle Web Agent to process the request. Id. Thus, a request for a dynamic page is "intercepted" at the Web Listener and passed to the Web Agent. With reference to Fig. 4 of the 554 patent reproduced above, the Web Listener of the Oracle system corresponds to the Web server/interceptor blocks of the claimed system. "The Oracle Web Agent is a CGI program that ... logs into the [Oracle 7 Server] database and executes stored PL/SQL procedures that have been specified as part of the URL [of the received request]." (User's Guide, p. 5-2; Montinola article, p. 3.) "A PL/SQL procedure extracts data from the Oracle 7 database and generates an HTML document .... " (User's Guide; p. 5-1; Montinola article, p. 4.) "The HTML output generated by a PL/SQL procedure is sent from the Oracle 7 Server to the Oracle Web Agent." Id. "The output is then passed on to

the Oracle Web Listener." Id. "The Web Listener sends the generated HTML document back to the Web client." Id. Thus, in the context of the 554 patent the Web Agent serves as a "dispatcher" to dispatch a request for a dynamic Web page to the Oracle 7 Server which, in turn, functions as a "page server." "For maximum performance the Web Listener is designed to run as an asynchronous engine ... providing high performance under a heavy load." (User's Guide, p. 3-2.) Indeed, as mentioned above, when the Web Listener receives a request for a dynamic Web page, it "spawns the Web Agent." (User's Guide, p. 5-4; Montinola article, pp. 3-4.) Inherent in the "asynchronous" Web Listener "spawning" the Web Agent, the Web Listener is released or freed to process other incoming requests. To facilitate identification of the particular data source from which to retrieve data from the Oracle 7 database, the Web Agent may pass a QUERY_STRING environment variable to a PL/SQL procedure. (User's Guide, p. 5-4.) Specifically, "[i]f the PL/SQL procedure requires parameters, these would have been passed to the Web Agent through the QUERY_STRING environment variable." Id. "The Web Agent would then pass these on to the PL/SQL procedure." Id. For example, part of a query such as "person=Gretzky" may be passed to a PL/SQL procedure using the QUERY_STRING environment variable. (User's Guide, p. 5-9.) The PL/SQL procedure will then use the parameter at least in part to identify the data to retrieve.

The Web Agent must "log on" to the Oracle 7 Server in order to interact with it. (User's Guide, p. 5-4.) In order to log on, "the Web Agent requires certain information, such as which server to connect to and what username and password to use." Id. Specifically, the

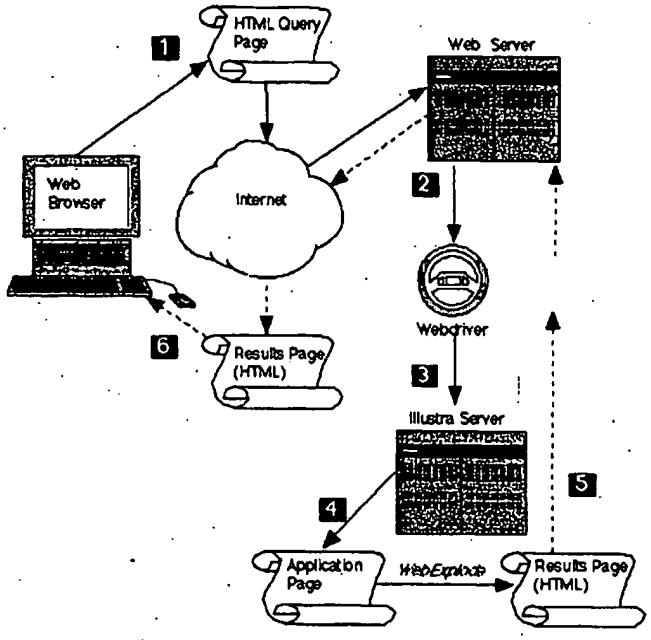
Web Agent "[e]xtracts the database service name embedded in the URL [of the received request] and logs on to the database using the userid/password defined by the service." (Montinola article, p. 4.) The Oracle WebServer also employs page caching to improve performance. Specifically, "[t]he Oracle Web Listener allows a configurable set of commonly accessed files to be cached in memory. This provides very good performance when these files are accessed by a client." (User's Guide, p. 3-2.) The Oracle WebServer also provides custom HTML extension templates to facilitate dynamic Web page creation. Indeed, "one of the main goals of the Oracle Web Agent is to eliminate the PL/SQL programmer's need to be intimately familiar with World Wide Web Technology. To this end, the Oracle WebServer includes a Developer's Toolkit made up of several PL/SQL packages that minimize a programmer's need to know HTML syntax .... [B]y using the Toolkit he or she will not need to hard code the exact syntax of HTML tags into PL/SQL procedures [i.e., the procedures that extract data from the database and insert the data into an HTML document]." (User's Guide, p. 6-1); (see also Montinola article, p. 3) (the "WebServer Developer's Toolkit ... ease[s] the generation of HTML tags from within the user's PL/SQL code"). In greater detail, "[t]he Oracle WebServer Developer's Toolkit includes ... Hypertext Procedures (HTP)." (User's Guide, p. 6-1.) "A hypertext procedure generates a line in an HTML document that contains the HTML tag that corresponds to its name. For instance, the htp.anchor procedure generates an anchor tag." Id. A Web developer uses these Hypertext Procedures in the PL/SQL code of a procedure designed to generate a dynamic Web page. There are a variety of Hypertext Procedures provided with the toolkit, including "Body Tags" that "are used in the main text of your HTML

page," "List Tags" that allow you to display information in lists, "Form Tags" that "are used to create and manipulate an HTML form," and "Table Tags" that "allow the user to insert tables and manipulate the size and columns of the table in a document." (User's Guide, pp. 6-1 to 6-31; also see, Montinola article, p. 3.) When a PL/SQL procedure retrieves data from the Oracle 7 database, it inserts the data into the Web page in accordance with the "template" defined by one or more of the Hypertext Procedures provided with the toolkit. For example, the "htp.tableOpen" and "htp.tableClose" Hypertext Procedures are used to insert retrieved data into an HTML table within a dynamic Web page. (User's Guide, pp. 6-29 to 6-30.) The above Hypertext Procedures constitute HTML extension templates that a Web developer can use to facilitate configuration of a dynamic Web page. In addition to the Hypertext Procedures provided as part of the toolkit, "[t]he design of the hypertext procedure ... package[] allows you to use customized extensions. Therefore, as the HTML standard changes, you can add new functionality similar to the hypertext procedure ... package[] to reflect those changes." (User's Guide, p. 6-37.)

John Gaffney, "Illustra's Web DataBlade Module, An Illustra Technical Paper," SIGMOD Record, Vol. 25, No. 1, March 1996 ("Illustra")

The Illustra reference discloses "a comprehensive toolset for creating Web-enabled database applications that dynamically retrieve and update Illustra database content."

(Illustra, p. 105.) The basic components and operation of Illustra's Web architecture is illustrated in Figure 2, reproduced below.



Illustra's architecture has three tiers of which the Web Browser represents the first tier, which corresponds to the client of the 554 patent; the Web Server is the second tier, which corresponds directly to the Web server of the 554 patent; and Illustra Server represents the third tier, corresponding to the page server of the 554 patent. In the Illustra architecture the Web Server offloads the processor-intensive creation of dynamic Web pages to the Illustra Server. The handling of a request for a dynamically generated Web page in the Illustra architecture is as follows. "1. The Web browser queries the Web server with a form submission request or other query." (Id., p. 110.) The Web Server receives the request from the Web browser. "2. The Web server launches the client database application," such as Webdriver. (Id.) The Web Server must examine the request

to determine that launching of the Webdriver is required. Therefore the Web Server inherently intercepts the request. As shown in the figure above, after the Webdriver has been launched, the request is routed to it. Additionally, after routing the request to the Webdriver, the Web Server is no longer occupied with the processing of the request and it is therefore released from its processing. "3. Based on a defined set of parameters, the driver connects to the specified database as a specified user, and then selects a specified Application page..." (Id.) Not only the Webdriver identifies the database, but it also connects to it as a specified user, which requires logging in. "4. WebExplode parses the extracted pages, executes the embedded queries, and then formats the results. This is accomplished within the database server." (Id. p. 110.) The Application page is a template. It is stored in the database and is populated with data retrieved from database. (See Id., p. 108.) The embedded queries are SQL statements, which when executed retrieve data from the databases. As discussed above, WebExplode inserts the dynamically retrieved data into the Application Page. "5. Based on the results of WebExplode, an Application page is placed in the database server memory space and returned to Webdriver..." "6. Webdriver... renders the Application Page through the Web server to the client browser program." The Illustra references also disclose that "the requests to the database are all managed and processed through these already-established connections."(id., p. 109.) Therefore, a connection cache is maintained.

Carl Lagoze, Erin Shaw, James R. Davis and Dean B. Krafft, "Dienst: Implementation Reference Manual," pp. 1-69 (May 5, 1995) ("Lagoze" and/or "Dienst")

Dienst discloses a protocol and server that provides distributed document libraries over the World Wide Web.

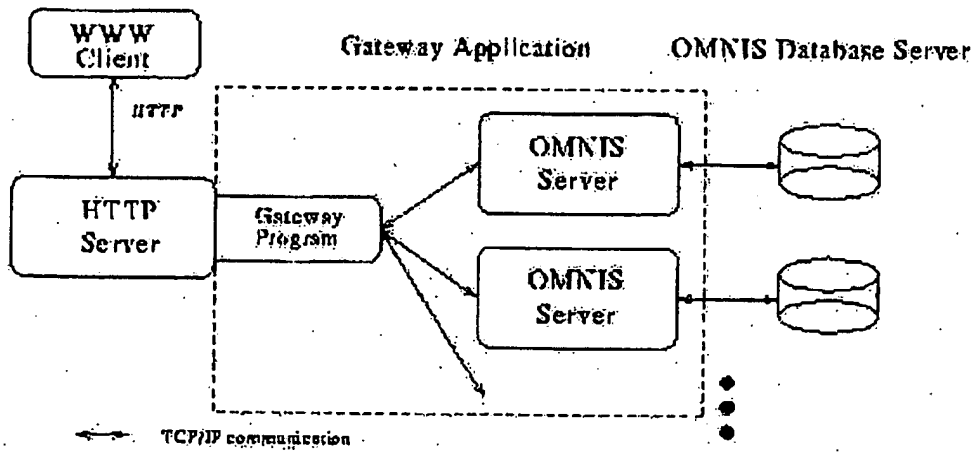
The Dienst Manual describes a system and method for managing requests from a WWW client via a WWW server for documents ("Web pages") stored on a page server (Dienst Server) or other data source such as a remote server. See generally Abstract, pages 6-11. The overall configuration is shown in Figure 2, at page 10.

Lagoze / Dienst includes a system and method for managing requests for documents ("Web Pages") stored on page server (Dienst server) or other server from WWW clients. The requester indicates that Dienst discloses the requests from the client being intercepted at the server by a CGI (Common Gateway Interface) stub as shown via figure 2 on page 10 of Dienst. The CGI stub dispatches the requests to a page server (Dienst server) which retrieves the requested dynamic content for delivery to the requesting client and which allows the WWW server to handle other incoming requests.

Alexander Clausnitzer, Pavel Vogel and Stephan Wiesener, "A WWW Interface to the OMNIS/Myriad Literature Retrieval Engine" (1995) ("Clausnitzer")

Clausnitzer describes a World-Wide-Web (Web) interface to a multimedia information search and retrieval system called "OMNIS/Myriad." The system is client/server-based

and enables a user with a Web browser to search and retrieve documents and information about those documents from a database. As shown in Figure 5 of the reference (reproduced below), one embodiment of the system architecture includes an HTTP server (i.e., Web server) that receives HTTP requests containing OMNIS/Myriad search requests from a user's Web client (i.e., Web browser), a CGI "Gateway Program," and one or more OMNIS servers, each connected to a respective database of documents. (§ 3.2.) "The CGI program passes the WWW client's requests to one of the OMNIS servers which translate them into OMNIS queries for the database. The query results are sent back to the CGI program which transforms it into a WWW client readable format," i.e., a Web page. (Id.) Because each search request may be different and will produce different search results, each such request from a Web client constitutes a dynamic Web page generation request to the Web server.



Again, the architecture of the system described and illustrated in the Clausnitzer reference is virtually identical to the partitioned architecture disclosed and claimed in the 554 patent, as a comparison of the figure above to Figure 4 of the 554 patent once again

demonstrates. As described in the Clausnitzer reference, when the Web server recognizes from the URL in an HTTP request received from the Web client that the request is an OMNIS query (i.e., a dynamic Web page generation request), the Web server interrupts its processing of the request and passes the request to the CGI Gateway Program. (See §3.3 and description accompanying Figure 6.) Thus, the request is intercepted. The CGI program serves as a dispatcher, dispatching the request to one or more OMNIS servers. (§ 3.2 and Figure 5.)

Inherent in the CGI program accepting the request and forwarding it to an OMNIS server, the Web server process is released from processing that request and can process other requests. Indeed, Clausnitzer notes that "[w]ith fast following HTTP requests the CGI program can be started several times by the HTTP server to run parallel," thus confirming that the HTTP server continues to process other requests after starting the CGI program for a prior request. (§ 3.3.) "Each OMNIS server has a permanent connection to a single database," which serves as the "data source" from which the OMNIS server dynamically retrieves query results. (§ 3.2.) The OMNIS server to which a particular request is dispatched will process the query and return the results. The OMNIS server thus constitutes a "page server." "The query results are sent back to the CGI program which transforms it into a WWW client readable format," i.e., a Web page, which is then transmitted back to the client. (§ 3.2.) Because "[e]ach OMNIS server has a permanent connection to a single database," the connection that a given OMNIS server has to its data source is necessarily, maintained or cached. Also, OMNIS servers do require login and logout (see, § 3.2). Thus processing an OMNIS query necessarily requires at least

one initial login to the OMNIS server that processes the request. In addition to the foregoing features, Clausnitzer also teaches the use of custom HTML extension templates: A major point in the realization of the OMNIS-HTTP gateway was to make it easy to customize the pages and forms for different types of OMNIS databases. Since many databases should be available via WWW the gateway application has to provide easy methods for the administrator to add and change database parameters. For these purposes a special Mask Definition Language was developed. The masks consist of the fixed HTML fragments [i.e., custom HTML extension templates] which are filled by the gateway application and returned to the client as request results." (§ 3.3.)

Christian Derler, "The World-Wide Web Gateway to Hyper-G: Using a Connectionless Protocol to Access Session-Oriented Services," Institut fur Informationsverarbeitung und Computergestutzte neue Medien, pp. 1-104 (March 1995) ("Derler")

The Derler Thesis describes a system and method for managing requests from a WWW (World Wide Web) client via a WWW server (WWW Gateway) for documents or other information stored on a page server (Hyper-G Server) or other data source such as a document cache or remote server. See generally Abstract, Introduction at pages 1-2, and chapter 4 describing the WWW Gateway, particularly at pp. 48-55. The WWW Gateway includes software features described at pages 48-55, including Master, Passing, Slave, and Child processes, which intercept HTTP requests from the WWW client at the WWW Gateway, and dispatch such requests to the page server (Slave Processes and Hyper-G Server), while the WWW Gateway processes other requests

Bowman et al., "Harvest: A Scalable, Customizable Discovery and Access System" (March 12, 1995) ("Harvest")

Specifically, Harvest teaches "a system that provides an integrated set of customizable tools for gathering information from diverse repositories, building topic-specific content indexes, flexibly searching the indexes, widely replicating them, and caching objects as they are retrieved across the Internet. The system interoperates with WWW clients and with HTTP, FTP, Gopher, and NetNews information resources." (Harvest, p. 1.) Of particular relevance to claim 4 of the 554 patent, Harvest teaches the use of a connection cache. For example, Harvest explains that "[b]ecause [remote data retrieval] poses so much more load on provider-site servers, we implemented a connection caching mechanism, which attempts to keep connections open across individual object retrievals." (Id, at p. 10.) Harvest also teaches a page cache as found in claim 6 of the 554 patent. Specifically, Harvest explains that "[i]n addition to caching Gopher, HTTP, FTP, and NetNews objects, we maintain a cache of recent DNS name-to-address mappings to optimize common-case cache behavior." (Id, at p. 19.)

Antchev et al., "A WWW Learning Environment for Mathematics" (December 12, 1995)("Antchev")

Antchev teaches a system for providing mathematic learning experiences over the Internet. (Antchev, p. 1.) To facilitate this, an "HTTP-server spawns [a] handler which is referred to by the URL of an incoming information request. The handler being invoked should return the appropriate HTML or other type of document or it should generate such a document on the fly." (Id. at p. 4.) Antchev, like Harvest, also teaches the use of a connection cache. For example, Antchev explains that because for "most of the exercises there might be several requests which form a session," a context tree data structure is used to store state information across an entire user session. Thus, Antchev maintains a "connection cache" across the session.

Ashley Beitz, Renato Iannella, Andreas Vogel, Zhonghua Yang, "Integrating WWW and Middleware"(May 2, 1995) ("Beitz")

Specifically, Beitz teaches an "architecture which enables WWW clients to access and invoke middleware services. In the architecture, the clients interact with the WWW, which in turn interacts with a range of middleware systems which provides access to the distributed system services." (Beitz, p. 2.) The Beitz reference is particularly relevant for its teaching of "logging into" a data source, as recited in claim 5 of the 554 patent. For example, Beitz teaches "filling in the necessary parameters to invoke [a] selected RPC operation." (Id. at p. 5.)

Ari Luotonen, and Kevin Altis, "World-Wide Web Proxies" (April 1994) ("Luotonen")

Specifically, Luotonen teaches a proxy server that "waits for a request from inside the firewall, forwards the request to [a] remote server outside the firewall, reads the response and then sends it back to [a] client." (Luotonen, p. 1.) Of particular relevance to claim 6 of the 554 patent is Luotonen's teaching of the use of a page cache. For example, Luotonen explains that "[a] brand new feature is [page] caching performed by the proxy, resulting in shorter response times after the first document fetch." (Id.)

Richard Knudson, "Application Development with Microsoft's Internet Information Server" (February 2, 1996) ("Knudson")

The Knudson reference describes Microsoft's "Internet Information Server" product that allows for the generation of dynamic Web pages. Knudson, p. 1. The reference is particularly relevant because it teaches the use of "custom HTML extension templates" as recited in claims 7 and 8 of the 554 patent. Specifically, pages 3-4 of Knudson show examples of custom HTML extension templates into which data retrieved by the Internet Information Server is inserted when generating a dynamic Web page.

IV. Background and Overview

USPN 5,894,554 Lowery et al.

The '554 Patent issued on April 13, 1999. The patent generally relates to handling Web page requests, managing Web sites, and dynamic Web page generation. (Abstract & Col. 2:15-23).

The patent distinguishes some Web pages as having a static nature that remains static until manually modified and other Web pages as being dynamic Web pages which contain content that is generated dynamically by retrieving the necessary requested data and generating the requested Web page dynamically. (Col. 1:38-55).

The patent is said to address a problem of managing numerous Web pages and Web page requests of Web clients at a Web site (Col. 2:2-12). With reference to Figure 4, reproduced below, the method is performed in a system including a Web server 201, separate page servers 404, and data sources 406, 408 and 410. The patent includes routing a Web request from the Web server to a Page server. Such routing is described as being performed by the dispatcher 402 (Col. 5 line 60 - col. 6 line 11). The routing frees the Web server to handle subsequent incoming requests. The Page server may then process the request, and the Web server is released to process other requests (Col. 2:20-35; Col. 4:54 - Col. 6:32). In this manner, dynamic Web pages may be generated by the Page servers.

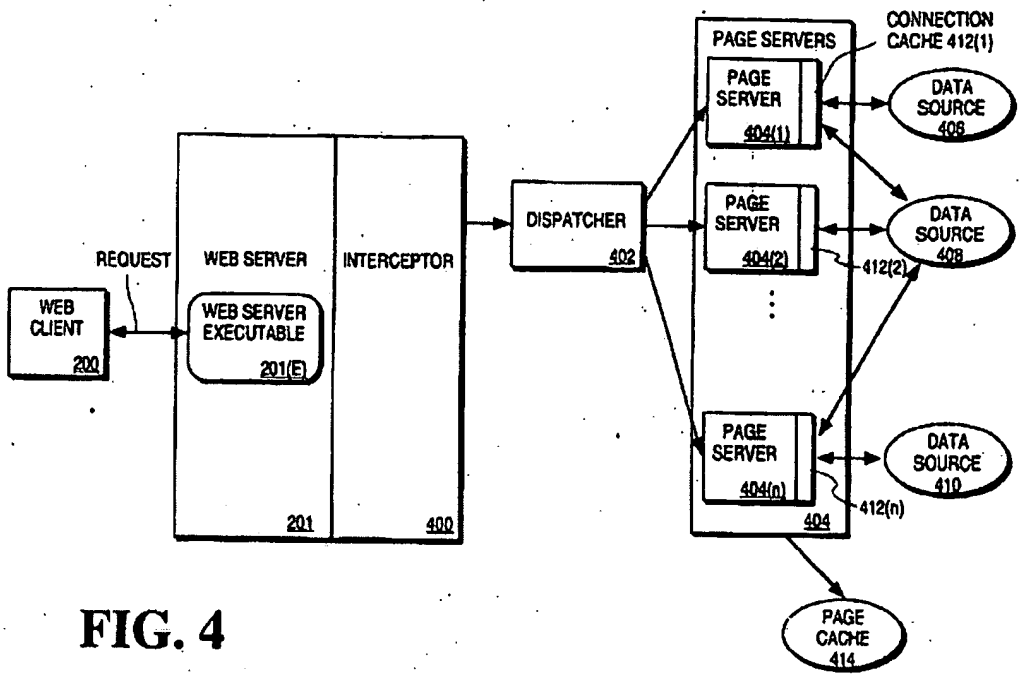


FIG. 4

A review of the prosecution history reveals that Examiner initially rejected claims (CTNF 07/03/1997) based on USPN 5,532,838 to Barbari and "Beyond the Web: Excavating the Real World Via Mosaic" by Goldberg et al. Subsequent arguments persuaded Examiner to apply new art. A second non final office action applied USPN 5,404,527 to Irwin et al., to reject claim limitations. Arguments overcame claim rejections.

An Interview (12/29/1998) resulted in claim amendments that cancelled original dependent claim 2, independent claims 13 and 14, and dependent claim 15. Original claim 2 limitations were rolled up into remaining independent claims. Resulting independent claims (final claim

numbering: claim 1-method claim, claim 9 - system claim, and claim 11- machine readable medium version) recite similar limitations including:

...routing said request...wherein said routing step further includes the steps of intercepting said request...routing said request from said Web server/ second computer to a dispatcher, and dispatching said request to said page server...

The examiner did not indicate reasons for allowance during prosecution of the application which became the 5,894,554 patent to Lowery. It appears that the combination of limitations, as a whole, was not found in the prior art of record.

At the time of the Interview (12/29/1998) new related patent prior art was presented on a Form 892, citing USPN 5,761,673 to Bookman et al. and USPN 5,751,956 to Kirsch.

Bookman discloses a method and apparatus for generating dynamic Web pages. Specifically, the present invention claims a method and apparatus for generating dynamic Web pages on a Web server by invoking and executing predefined procedural packages stored in a database. The claimed invention receives an object request on the Web server and activates a Web agent on the Web server based on the object request. The Web agent invokes and executes the predefined procedural package to retrieve data from a data repository, and then formats the retrieved data as HTML output. (Abstract)

While Bookman appears to be relevant prior art, Bookman fails to fairly disclose "routing said request from said Web server to a page server", thereby freeing up the Web server to receive subsequent requests.

Kirsch discloses a Web server computer system which provides for server based controlled management over a client reference to a resource locator independently selected by a client computer system and referencing a server external Web server. The Web server system provides a client system with a predetermined URL reference to the Web server system encoded with predetermined redirection and accounting data including a reference to a second server system. On receipt by the first Web server system of the predetermined URL reference from the client system, the predetermined redirection and accounting data is decoded from the predetermined URL and processed by the Web server system to provide the client system with a redirection message including the reference to the second server system. The accounting data is processed by the Web server system and resulting data is selectively stored by the Web server system. (Abstract) Kirsch discloses a Web server that receives a client reference (request), tracks (accounting data) the client activity, and provides the client with a redirection message (col. 5: 37), "thereby minimizing the CPU and disk intensive load on the Web server computer system and the resulting latency (col. 5: 50-51) (i.e. client system is used to redirect to second server system)."

In contrast, the Lowry patent, '554, passes the client request from the Web server through a dispatcher, to a page server (second server) to minimize the CPU and disk intensive load on the

Web server. While Kirsch appears to be relevant prior art, Kirsch fails to fairly disclose a Web server that routes said request through a dispatcher, to a page server.

Both '554 and '956 patents attempt to reduce the workload of the Web server, but do it in different ways.

V. Claim Rejections – Relevant Statutes and Detailed Analysis

35 USC § 102

The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (a) the invention was known or used by others in this country, or patented or described in a printed publication in this or a foreign country, before the invention thereof by the applicant for a patent.

Claims 1-11 are rejected under 35 U.S.C. 102(a) as being anticipated by Derler.

Claim 1.

[a] A computer-implemented method for managing a dynamic Web page generation request to a Web server, said computer-implemented method comprising the steps of:

The Derler Thesis discloses a computer-implemented method for managing dynamic Web page generation requests from a WWW client sent to a Web server (WWW Gateway). Introduction, pp. 1-2; § 3.3.4 at p. 37, § 4.1 at pp. 43-44; § 4.7.2 at pp. 94-96; Figs. 4.16-4.17 at p. 95. The WWW Gateway manages dynamic Web page generation requests from remote WWW clients

and thus constitutes a Web server. Operation of the WWW Gateway is a computer-implemented method, see generally Chapter 4.2 2. Derler's WWW clients use Web browsers to access information from the Hyper-G servers, therefore the Derler system is handling dynamic "Web page" generation requests. See Derler at p. 4 ("WWW clients, often called 'WWW browsers'"); p. 48; pp. 67-68 (explaining user interaction with Hyper-G systems with "WWW clients like Mosaic or Netscape", both of which are Web browsers).

[bl] routing said request from said Web server to a page server,

Derler: Web page requests are directed from the WWW Gateway (the "Web server") to the Hyper-G Server ("page server"). (§ 4.2.2 at pp. 48-49, § 4.2.3 at p. 50, 52, § 4.7.2 at p. 95, Fig. 4.17.) As described in Section 4.2.3, and accompanying Figures 4.1 - 4.7, a request from a client is initially received by a "Master Process" within the Web server. When the request is received, the Master Process "forks", generating a child process to process the request. "The master can close its connection to the client immediately after forking and is then ready to accept the next connection." If this is the first request from the Web client, the child process becomes a "Slave Process" (route directly to Slave Process / Page Server) which opens a connection to, and starts a session with, the Hyper-G server on behalf of the Web client. The Slave Process then sends a standard "entry page" back to the Web client, enabling the user to begin searching the Hyper-G server database. Subsequent search requests from the Web client are then received by the Master Process, which again forks to create a child process and to release the Master Process to process other requests. For these subsequent requests, the child process becomes a "Passing Process"

(route request from Web Server to Passing Process) that looks for the previously created Slave Process (page server) that holds the open connection to the Hyper-G server and then passes (dispatches) the client request to that Slave Process. The Slave Process will then transform the request into a Hyper-G request and send it to the Hyper-G server. The Hyper-G Server's response is transmitted back to the Slave Process. The Slave Process then transforms the response, as needed, and passes a Web page containing the results back to the Passing Process which relays it the Web client. In that respect, the Slave Process functions as a "page server," and the Hyper-G server functions as a data source. Thus, the system routes a client request from the Master Process of the Web server to a Slave Process that serves as a "page server" for handling the request.

[b2] said page server receiving said request and

As described in Derler, when the Master Process "forks" to pass a client request to a Slave or Passing Process, "the master can close its connection to the client immediately after forking and is then ready to accept the next connection." (§ 4.2.3.) Derler: The client request is received by the slave process (page server) for processing. (See Id.)

[b3] releasing said Web server to process other requests,

Derler's WWW Gateway uses parallel processing of Web page requests wherein the WWW Gateway is released from one request to process other requests. See § 4.2.2 at pp. 48-49; § 4.2.3

at p. 49 (Master process of WWW Gateway), Figs. 4.1, 4.2, 4.3, 4.4 and 4.5 (pp. 52-55) illustrating process of releasing WWW Gateway to process other requests in parallel. Derler expressly recognizes that to avoid problems with throughput, "the process which accepts connections from the "outside" [i.e., from a Web client] is not burdened with the actual processing of the request, but instead is allowed to accept further requests while others are still being processed." (§ 4.2.2 -"Parallel Processing of Requests".)

[b4] wherein said routing step further includes the steps of intercepting said request at said Web server,

Derler: (§ 4.2.2 at pp. 48-49; § 4.2.3 at pp. 49-50; Figs. 4.16-4.17 and accompanying disclosure at pp. 94- 95); function of Master Process executing on WWW Gateway is to intercept Web page requests. Forked process is intercepted by a Passing Process.

[b5] routing said request from said Web server to a dispatcher, and

Derler: The Web Server routes the request to the Passing Process which in turn dispatches the request to the Slave Process (Page Server) that holds a connection to the Hyper-G server. Passing Process identifies appropriate connection. (§ 4.2.2 at p. 48; § 4.2.3 at p. 50, Figs. 4.16-4.17 and accompanying disclosure at pp. 94-95) (depicting GET request path, including WWW Gateway server to Hyper-G server); Web Server (Master) routes request to dispatcher (Passing

Process is dispatcher/dispatches to identified Slave Process / Page Server), which looks for session keys to determine where to handle Web page request;

The Passing Process then dispatches the request to the Slave Process (page server) that holds an open connection to the Hyper-G server for requests from that particular client. As shown in Figures 4.1- 4.7, there may be many Slave Processes, and the Passing Process (dispatching the request to an identified slave process) must identify the appropriate one. The Passing Process thus functions to dispatch the request to the appropriate Slave Process.

[b6] dispatching said request to said page server;

Derler: Id.; The Passing Process then dispatches the request to the Slave Process (page server) that holds an open connection to the Hyper-G server for requests from that particular client. As shown in Figures 4.1- 4.7, there may be many Slave Processes, and the Passing Process (dispatching the request to an identified slave process) must identify the appropriate one. The Passing Process thus functions to dispatch the request to the appropriate Slave Process. Figure 4.1, page server is Slave process which opens a connection to Hyper-G server (data source) and sends Hyper-G request to Hyper-G server. See Fig. 4.5 – 4.6, Slave A and Slave B process the request (page servers).

The Hyper-G server as stated in § 3.5 of Derler may perform a further requesting from a remote page server. The reference discloses a Hyper-G page server obtaining information from remote

servers; the Hyper-G server fetches the information [from a remote server] and delivers it to the client, via a Slave Process (§ 4.4.2)

[c] processing said request, said processing being performed by said page server while said Web server concurrently processes said other requests; and

Derler: § 4.2.2 at pp. 48-49; § 4.2.3 at p. 49; pp. 52-54, processing of request by Hyper-G server; Figs. 4.3, 4.4, 4.5, such processing occurs concurrently with the WWW Gateway processing other requests. See also § 4.2.3 at p. 50 ("When it [a slave process on the WWW Gateway] has accepted a connection initiated by the passing process, it reads the request, processes it (usually by transforming it into a Hyper-G request for the server, receives the response from the Hyper-G server, transforms the response again if necessary, sends it to the passing service, closes the connection to the passing process.") Necessarily, since the client receives a response from the Hyper-G server, the Hyper-G server has processed the request. The slave process connects to Hyper-G server page server for processing of the request. See § 4.4.2, "The class WWWSlave contains all data and code necessary to process requests…"

As described in Derler, when the Master Process "forks" to pass a client request to a Slave or Passing Process, "the master can close its connection to the client immediately after forking and is then ready to accept the next connection (web server concurrently processes said other requests)." § 4.2.3. Indeed, Derler expressly recognizes that to avoid problems with throughput, "the process which accepts connections from the "outside" [i.e., from a Web client] is not

burdened with the actual processing of the request, but instead is allowed to accept further requests while others are still being processed." § 4.2.2 -"Parallel Processing of Requests".

[dl] dynamically generating a Web page in response to said request,

Derler: § 3.3.4 at p. 37; Fig. 4.10 at p. 71 (depicting sample of a dynamically-generated Web page in response to a search request). The Slave Process receives the results of the search from the Hyper- G server's database and generates a responsive Web page for the client, § 4.4.2 See, e.g., § 4.5.5 & Figure 4.13 (illustrating an exemplary Web page displaying the results of a search).

[d2] said Web page including data dynamically retrieved from one or more data sources.

Derler: Id.; data source for Web page can be data from a data cache, see § 3.3.5; see also p. 95, showing data retrieved from a remote data source in the form of a remote server.

Claim 2.

The computer-implemented method in claim 1 wherein said step of processing said request includes the step of identifying said one or more data sources from which to retrieve said data.

Derler: § 4.7.2, Fig. 4.17 (p. 95) (Hyper-G server identifies remote server from which to retrieve data); see also § 4.1 (integration of WWW databases into Hyper-G, inherently such databases

would be identified so as to retrieve data from such WWW databases). See also § 3.2.3 ("Supporting bidirectional links can easily be done by keeping links and anchors in a separated database, called the link server. This link server is an object oriented database. It holds information about Hyper-G "objects', which can be links, anchors, descriptions of documents, relations between such objects."); § 3.5 ("should information from a remote server be needed, the Hyper-G server which the client is connected to fetches the information and delivers it to the client.").

Claim 3.

The computer-implemented method in claim 2 wherein said step of dynamically generating said Web page includes the step of dynamically retrieving said data from said one or more data sources.

Derler: § 3.3.4 at p. 37; Fig. 4.10 at p. 71 (depicting sample dynamically-generated Web pages; data is retrieved dynamically from one or more sources); § 3.3.5 at p. 37 (dynamically retrieving data from document cache).

Claim 4.

The computer-implemented method in claim 3 wherein said step of processing said request includes the step of said page server maintaining a connection cache to said one or more data sources.

Derler: § 3.5 at p. 40 (attributes and links to documents stored in data sources are stored separately in a database, and as such constitute a connection cache).

Alternatively, a Slave Process receives a first request from a particular Web client, opens a connection to, and starts a session with, the Hyper-G server on behalf of the Web client. It maintains that connection to the Hyper-G server so that it can service all subsequent requests from the same client. Thus, the Slave Process caches the open connection to the Hyper-G server. The purpose of this is to avoid having to reestablish connections to the Hyper-G server each time, § 4.2.3.

Claim 5.

The computer-implemented method in claim 3 wherein said step of processing said request includes the step of logging into said one or more data sources.

Derler: § 3.3.3 at pp. 35-36 (access rights dictated by username and password, inherently teaches "logging in" to data source; § 4.5.6.1 at pp. 76-77; Figure 4.8 ("you have been logged into the IICM Information Server").

Claim 6.

The computer-implemented method in claim 3 wherein said step of dynamically generating said Web page includes the step of maintaining a page cache containing said Web page.

Derler: § 3.3.5 at p. 37 (caching of documents (Web pages) in a cache; step of dynamically generating a Web page is performed by retrieving a cached document in response to a request for the document).

Claim 7.

The computer-implemented method in claim 3 wherein said page server includes custom HTML extension templates for configuring said Web page.

Derler: § 3.3.4 at p. 37 (Hyper-G server generates HTML elements (custom extension templates) configuring for use search results in form of a Web page; see generally § 4.5 and subsections; §§ 2.5, 2.5.4 (fill-out forms are "extensions of initial HTML specification"), §§ 4.6.1 (HTML files provide user interface), 4.6.1.1- 4.6.1.2; 4.6.

Claim 8.

The computer-implemented method in claim 7 wherein said step of processing said request further includes the step of inserting said dynamically retrieved data from said one or more data sources into said custom HTML extension templates.

Derler: Id., e.g., § 4.6.1.1; Figure 4.13 shows data retrieved from data source inserted into HTML extension template defining presentation of results of search.

Regarding Claim 9:

With regard to the limitations of claims under reexamination, an analysis must be made under 35 USC 112, sixth paragraph.

According to **MPEP 2181**:

A claim limitation will be presumed to invoke 35 U.S.C. 112, sixth paragraph, if it meets the following 3-prong analysis:

(A) the claim limitations must use the phrase "means for" or "step for;"

(B) the "means for" or "step for" must be modified by functional language; and

(C) the phrase "means for" or "step for" must not be modified by sufficient structure, material, or acts for achieving the specified function.

Such a claim limitation is to be construed to cover the corresponding structure, described in the specification and equivalents thereof.

MPEP 2183 Making a Prima Facie Case of Equivalence

Factors that support a conclusion that the prior art element is an equivalent are:

(A) the prior art element performs the identical function specified in the claim in substantially the same way, and produces substantially the same results as the corresponding element disclosed in the specification.

(B) a person of ordinary skill in the art would have recognized the interchangeability of the element shown in the prior art for the corresponding element disclosed in the specification.

(C) there are insubstantial differences between the prior art element and the corresponding element disclosed in the specification.

(D) the prior art element is a structural equivalent of the corresponding element disclosed in the specification. That is, the prior art element performs the function specified in the claim in substantially the same manner as the function is performed by the corresponding element described in the specification.

In reference to USPN 5,894,554 to Lowery et al:

The cited function, “generating said request”, is linked to a corresponding structure, "means for generating said requests."

The "means for generating a request" is described as a Web client machine running a Web browser. (Lowery: Col. 1:24-26; Col. 2:4-7; Col. 4:12-15; Col. 4:55-57; Col. 6:27-31; See Col. 8:25-29.) The Web Client Computer (the corresponding structure), as noted in the Specification, having a processor which operates a Web browser, is the corresponding structure that accomplishes the function of generating a request. (a processor of a computer that is, or has, a Web client running a Web browser" or equivalents thereof).

The cited function, "receiving said request from said first computer", is linked to a corresponding structure, the "means for receiving said request from said first computer."

The "means for receiving said request from said first computer" is described by the Specification as the Web server 201 receiving requests from the Web client 200. (Lowery: Figure 4; Figure 5; Col. 4:54-59; See Figure 2; Col. 3:64-Col. 4:10.) The Web server 201 is also repeatedly described as including Web server executable, Id. As described above, the Specification also establishes that that even if a Web server is software, the software module operates on a computer as described within the specification. The corresponding structure of the "means for receiving" is "a processor of a computer that is, or has, a Web server running Web server executable" or equivalents thereof. The corresponding structure accomplishes the function of "receiving said request from said first computer."

The cited function, "processing said requests and dynamically generating a web page in response to said request" is linked to a corresponding structure, a "page server processing means processing said requests and dynamically generating a web page in response to said request."

With regard to the function of the "processing said requests and dynamically generating a web page in response to said request", the Specification shows that such function is accomplished by a corresponding Page server 404(1)-(n) structure. (Lowery: Figure 4; Figure 5; Col. 5:37-Col. 6:31; Col. 8:39-43.) The Page server is page generation software. The page server software is operated on a computer system, a processor of a computer that runs Page server software/ page-

commit=/storefront/commit session
rollback=/storefront/rollback session
[TX2]
[EMPLOYEE]
name=EMPLOYEEACCOUNTS
belong-to-list=/EMPLOYEE
/BANKING
resource-list=/PERSONNEL
/BANK1
begin=/employee/open session
commit=/employee/commit session
rollback=/employee/rollback session

For each type of transaction the metadata includes various attributes. According to one embodiment, the attributes include a cartridge name, a transaction name, a belong-to-list, a resource-list, begin, commit and rollback TRANSACTION URLs. In the example given above, the cartridge name for TX1, is STOREFRONT, the transaction name is STOREACCOUNTS, the belong-to-list consists of /STOREFRONT and /BANKING, the resource-list consists of /SEARS and /BANK1, the begin transaction URL is /storefront/open session, the commit transaction URL is /storefront/commit session and the rollback transaction URL is /storefront/rollback session.

The cartridge name attribute identifies the particular type of cartridge that the dispatcher communicates with to perform the operations of the transaction. The transaction name attribute uniquely identifies the type of transaction relative to other transaction types. The belong-to-list of a transaction type lists the cartridges that may participate in the performance of the transaction. The resource-list is the list of resources that are affected by the performance of transactions that are of the transaction type. The begin transaction URL is the URL that signals that a transaction of this type is about to begin. The commit transaction URL is the URL that signals that a transaction of this type that is currently in progress should be committed. The rollback transaction URL is the URL that signals that a transact on of this type that has already started should be rolled back. How each of these attribute values is used during the performance of a transaction shall be described in greater detail below.

TRANSACTION OVERVIEW

FIG. **6** is a block diagram of a system **600** that provides for the processing of multiple-request transactions in a stateless environment according to one embodiment of the invention. FIG. **6** is similar to FIG. **2** and therefore like components have been numbered alike. Within this document, the term browser request and the term transaction request are used interchangeably. The term multiple-request transaction is used to refer to a single transaction that is comprised of two or more browser requests.

As described earlier, cartridge execution engine **228** communicates with a plurality of dispatchers (e.g. one or more of dispatchers **214**, **220** and **226**) through object request broker **282** to receive browser messages. These browser messages may be sent from a plurality of browsers connected to the Internet **208**. In addition to the plurality of dispatchers, cartridge execution engine **228** also communicates with a cartridge **230**, configuration provider **256** and transaction manager **606**. As previously described above, the cartridge **230** represents a module of code that is either 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. The combination of cartridge execution engine **228**, transaction manager **606** and cartridge **230** constitute a cartridge instance.

A particular cartridge may be associated with a plurality of database servers for access to a plurality of databases. In this example, cartridge **230** has the ability to process database transactions according to the Structured Query Language (SQL) by accessing database **610** and database **614** through database server **608** and database server **612** respectively.

Transaction manager **606** represents a coordinating module that is associated with cartridge execution engine **228** and functions to coordinate the execution of multiple-request transactions in the stateless web environment. In coordinating the execution of multiple-request transactions, transaction manager **606** retains no state information for the multiple-request transactions. The transaction manager **606** communicates with cartridge execution engine **228** to receive transaction control messages. Using the information contained in the transaction control messages, the transaction manager **606** interacts with database servers **608** and **612** to cause changes made during multiple-request transactions to respective databases **610** and **614** to be either committed or rolled back as an atomic unit of work.

IDENTIFYING TRANSACTIONS

Browser requests that are associated with multiple-request transactions include a globally unique transaction ID. The globally unique transaction ID within a browser request is used to identify the multiple-request transaction to which the browser request belongs. According to one embodiment, when a browser request is received that contains a begin transaction URL, the transaction manager creates a globally unique transaction ID. This globally unique transaction ID is returned to the sending browser, and is sent by the browser in subsequent browser requests that are associated with the same multiple-request transaction.

In certain embodiments, when returned to the browser from which a multiple-request transaction was initiated, the globally unique transaction ID associated with the particular multiple-request transaction is stored as cookie information on the client executing the browser. When a subsequent browser request is sent by the browser, the dispatcher determines if the subsequent request contains a begin transaction URL. If the request does not contain a begin transaction URL, then the dispatcher obtains the globally unique transaction ID associated with the browser request by reading the sending browser's cookie information using the HTTP protocol standards.

For example, when a browser **202** sends a first browser request associated with a transaction and a begin transaction URL, the transaction manager **606** creates a unique browser identifier and sends it to the dispatcher **214**. The dispatcher **214** then causes the globally unique transaction ID to be stored as cookie information on browser **202**. When browser **202** sends a second browser request that is associated with the same transaction, the dispatcher **214** obtains the globally unique transaction ID contained in the cookie information of browser **202**.

Using the globally unique transaction ID, the database servers that ultimately process the browser request can determine that both the first browser request and the second browser request are associated with the same multiple-request transaction. Because a particular browser may be executing more than one transaction at a time, in certain embodiments, the cartridge name for the particular transaction is contained within each globally unique transaction ID and is used to help identify the particular transaction to which the globally unique transaction ID corresponds.

ORCL01622490

In certain situations cookie information may not be available on a particular browser. For example, a particular browser may not support the use of cookies or a particular user may choose to deny access to the browser cookie information. Therefore, in certain embodiments, the transaction identifiers are embedded in the messages returned to a browser, and sent out by the browser in subsequent browser requests. This can be accomplished by annotating the URLs that are associated with the hyperlinks of the HTML page that is returned to the browser **202**. Based upon the globally unique transaction ID that is sent out as part of the browser request URL, the database servers that ultimately perform the operations specified in the browser requests can use the globally unique transaction ID to identify the multiple-request transaction to which each particular browser request belongs.

TRANSACTION CARTRIDGE INSTANTIATION

Each browser request contains URL information that is sent from the sending browser in response to a user of the browser selecting a hypertext link on an HTML page. The URL information includes a Uniform Resource Indicator (URI) portion and a header section. The URI portion includes transaction state information and a cartridge name. The transaction state information is used to identify the particular state of a multiple-request transaction. The cartridge name is used to identify the cartridge type and allows the cartridge execution engine to identify the metadata that is associated with the browser request.

The header section is used to store a globally unique transaction ID that is used by the database servers to identify the multiple-request transaction that is associated with a particular transaction request.

When a listener receives the browser request, it passes the browser request to the dispatcher. The dispatcher then communicates with the virtual path manager to determine the cartridge type that is associated with the browser request. In one embodiment, the dispatcher forwards the information contained in the URI to the virtual path manager. Using the information in the URI, the virtual path manager communicates with the configuration provider to determine the cartridge type that is associated with the browser message.

Once the cartridge type is identified, the virtual path manager returns data that identifies the cartridge type to the dispatcher. The dispatcher then searches a cartridge instance pointer list that includes pointers to cartridge instances that have previously been associated with the particular dispatcher. If the dispatcher locates a pointer to a cartridge instance that is of the cartridge type that is associated with the browser request, the dispatcher uses the pointer to send a revised browser message to the cartridge instance.

If the dispatcher does not locate a pointer to the type of cartridge instance that is associated with the browser request, the dispatcher communicates with the resource manager to obtain a cartridge instance of that type. In obtaining the cartridge instance, the dispatcher sends a message to the resource manager that includes the cartridge type that was previously identified by the virtual path manager.

Upon receiving the dispatcher message, the resource manager determines if a cartridge instance of the request type is available for use by searching a cartridge instance pointer table. If a cartridge instance pointer of the requested type is located in the cartridge instance pointer table, the resource manager sends a pointer to the available cartridge instance back to the dispatcher.

However, if a cartridge instance of the requested type is not available, the resource manager causes a cartridge instance of the request type to be instantiated. In one embodiment of the invention, the resource manager causes a cartridge instance of the requested type to be instantiated by requesting a particular cartridge factory process to create a cartridge instance of the request type. Cartridge factory processes may be located across multiple machines. When a particular cartridge factory process is requested to instantiate a cartridge instance, it instantiates the cartridge instance on the same machine that the cartridge factory is currently executing on. Therefore, the resource manager selects which cartridge factory to use based on the particular machine the resource manager chooses to instantiate the cartridge instance.

Upon receiving a request to instantiate a cartridge instance, the cartridge factory process instantiates an instance of a cartridge execution engine. Once the cartridge execution engine is instantiated, the cartridge execution engine obtains the transaction information, if any, that is associated with the requested cartridge type. For example, if the requested cartridge type is of type STOREFRONT as described in TX1 above, the cartridge execution engine obtains and stores the metadata information that is associated with TX1. This metadata information is used by the cartridge instance to process transactions.

After obtaining the metadata information, the cartridge execution engine instantiates a cartridge of the requested cartridge type. The instance of the cartridge that is created is dynamically linked with the cartridge execution engine. The cartridge execution engine then instantiates a transaction manager. The transaction manager instance is dynamically linked with the cartridge and the cartridge execution engine to form a cartridge instance.

Once the cartridge instance is formed, the transaction manager uses the metadata information that was previously stored by the cartridge execution engine to open connections with the databases that were identified in the resource-list of the metadata. These connections are retained by the transaction manager and later used to provide database handles to the associated cartridge and to control the processing of multiple-request transactions. For example, if the requested cartridge type is of type STOREFRONT, the resource-list is associated with a SEARS and BANK1 database. Using the resource-list information, the transaction manager opens a connection with the SEARS database and the BANK1 database by respectively establishing connections with the database servers associated with the SEARS and BANK1 databases. These connections are retained by the transaction manager and are used for processing transactions of type TX1.

After the transaction manager establishes its connections with the appropriate databases (i.e. through the database servers associated with the appropriate databases), the cartridge execution engine notifies the cartridge factory that a cartridge instance has been instantiated by returning a pointer to the cartridge instance back to the cartridge factory. Upon receiving the cartridge instance pointer, the cartridge factory sends the cartridge instance pointer to the resource manager.

The resource manager then registers the cartridge instance pointer into its cartridge instance pointer table. The resource manager then sends the cartridge instance pointer to the dispatcher. Upon receiving the cartridge instance pointer from the resource manager, the dispatcher stores the cartridge instance pointer into its associated cartridge instance

ORCL01622491

pointer list. The dispatcher then uses the cartridge instance pointer to send a revised browser message to the cartridge instance.

CREATING REVISED BROWSER MESSAGES

Upon obtaining a cartridge instance pointer, the dispatcher creates a revised browser message using the information associated with the browser request. This revised browser message includes the URI, header information, the cartridge type and a dispatcher pointer that allows messages to be returned to the dispatcher. For example, a revised message for a transaction of type TX1 as described above, may include the following information:

URI=/storefront/open_session

header=NULL

cartridge name=[STOREFRONT]

dispatcher pointer=address XXXXX

In this example, the URI is a begin transaction URI (a URI that is used by the cartridge execution engine to identify the beginning of a multiple-request transaction). Because the URI is a begin transaction URI, a globally unique transaction ID has not yet been associated with the multiple-request transaction. Hence, the header that would contain the transaction ID is set to NULL. For ongoing multiple-request transactions (i.e. when the browser request does not contain a URI of /storefront/open_session)and in which cookies are used to store the globally unique transaction ID, the header will contain the unique transaction ID. This unique transaction ID allows the database servers to associate a transaction request with an ongoing multiple-request transaction

The cartridge name identifies the cartridge type and is used by the cartridge execution engine to identify the metadata that contains information about the transaction type associated with the particular browser request. In this example, the cartridge name of STOREFRONT identifies the metadata associated with TX1 as being associated with the browser request.

After creating the revised browser message, the dispatcher uses the previously obtained cartridge instance pointer to send the revised browser message to the cartridge instance. When the cartridge instance receives the revised browser message, the cartridge instance uses the cartridge type information to identify the metadata that is associated with the browser request. After identifying the metadata, the cartridge execution engine uses the URI information to determine the state of the transaction associated with the browser request.

For example, it shall be assumed that the browser request included a URI of "/storefront/open_session" and a cartridge type of STOREFRONT.

By looking at the metadata associated with the cartridge type of STOREFRONT (i.e. the metadata described in TX1 above), the cartridge execution engine **228** determines that the URI of /storefront/open_session corresponds to a "begin" transaction state. Using this same mechanism, the cartridge execution engine **228** can determine that a browser request containing a URI of /storefront/commit_session corresponds to a "commit" transaction state and that a browser request containing a URI of /storefront/rollback_session corresponds to a"rollback" transaction state.

In the case where the URI does not include a particular state (i.e. a URI consisting only of /storefront), the cartridge execution engine **228** assumes that the browser request is associated with an ongoing multiple-request transaction that is not ready to be either committed or rolled backed.

When the cartridge execution engine receives a revised browser message that is not associated with a "begin" transaction, the cartridge execution engine checks the header to determine if it specifies a globally unique transaction ID. If the header specifies a globally unique transaction ID, then cookie information was used to store the globally unique transaction ID. If the header does not specify a globally unique transaction ID, the cartridge execution engine then searches the URI to identify the globally unique transaction ID that is associated with the browser request. Once the cartridge execution engine locates the globally unique transaction ID, the cartridge execution engine includes the transaction ID in the transaction control messages that are sent to the transaction manager. The transaction manager then uses the globally unique transaction ID in communicating with the associated database servers to cause multiple-request transactions to be either committed or rolled back as an atomic unit of work.

PROCESSING TRANSACTIONS

FIG. 7A through 7I are a flow diagram illustrating a method for processing multiple-request transactions in a stateless environment according to an embodiment of the invention.

At step **702**, a revised browser message that was directed to cartridge **230** is intercepted by cartridge execution engine **228**. For the purposes of explanation, it shall be assumed that the revised browser message was sent by dispatcher **214** and that the revised browser message is associated with transaction TX1 as described above.

At step **704**, cartridge execution engine **228** determines if the revised browser message is associated with a transaction. If the revised browser message is not associated with a transaction, at step **706**, cartridge execution engine **228** forwards the revised browser message to cartridge **230** for cartridge **230** to perform the requested non-transactional finctions associated with the revised browser message. Once the cartridge performs the requested non-transactional functions, control returns to step **702** in order for the cartridge execution engine **228** to intercept the next revised browser message.

Otherwise, if the revised browser message is associated with a transaction, at step **708**, cartridge execution engine **228** determines the state of the transaction by first determining whether the revised browser message is associated with a begin transaction URI. In determining whether the revised browser message is associated with a begin transaction URI, the cartridge execution engine **228** uses the cartridge name to identify the previously stored metadata that includes the transaction attributes of the transaction type identified in the revised browser message. Using the previously stored metadata, the cartridge execution engine **228** determines if the revised browser message is associated with a begin transaction URI.

For example, it shall be assumed that the revised browser message contained a cartridge name of STOREFRONT and a URI of /storefront/open_session. Using the STOREFRONT cartridge name, the cartridge execution engine **228** determines that the revised browser message is associated with the metadata for transaction TX1. Using this metadata, the cartridge execution engine **228** determines that the URI of /storefront/open_session is associated with a begin transaction.

If the cartridge execution engine **228** determines that the revised browser message is not associated with a begin transaction, then control proceeds to step **744**.

If the cartridge execution engine **228** determines that the revised browser message is associated with a begin

ORCL01622492

transaction, then at step **712**, the cartridge execution engine **228** includes a begin transaction identifier (tx_begin) in a transaction control message. The cartridge execution engine **228** then sends the transaction control message to the transaction manager **606**.

At step **714**, upon receiving the begin transaction identifier, the transaction manager **606** creates a globally unique transaction ID that is used to identify subsequent browser requests that are associated with this multiple request transaction. In certain embodiments of the invention, the transaction ID is formed using the browser IP address, the transaction name and a particular timestamp value.

At step **716**, the cartridge execution engine **228** sends an operation message to cartridge **230** that is formed from information that is contained in the revised browser message. The operation message also includes a dispatcher pointer that identifies the dispatcher that sent the revised browser request (dispatcher **214**). This pointer allows the cartridge **230** to write information back to the dispatcher. At step **718**, upon receiving the operation message, the cartridge **230** sends a message to the transaction manager **606** requesting handles for access to the databases that are associated with the transaction.

At step **720**, transaction manager **606** returns handles to the appropriate database servers to allow the cartridge **230** to process the transaction request. For example, assuming database **610** is associated with the SEARS database and database **614** is associated with the BANK1 database, transaction manager **606** will return handles to database server **608** and **612** respectively.

At step **722**, cartridge **230** uses the handles returned from transaction manager **606** to execute the operations identified in the operation message that was sent by the cartridge execution engine **228**.

At step **724**, the cartridge **230** determines whether the sending browser allows cookie information to be associated with the browser. If the browser does not allow for cookie information to be associated with the browser, at step **726**, the cartridge **230** causes the hyperlinks of the HTML page that was generated in response to executing this transaction request to be annotated to include the globally unique transaction ID. By annotating the hyperlinks of the HTML page, the URIs contained in subsequent browser request will contain the globally unique transaction ID.

At step **728**, the cartridge **230** uses the dispatcher pointer to return back to the dispatcher **214** the HTML page that was generated in response to executing the transaction request. The cartridge **230** then notifies cartridge execution engine **228** that execution of the transaction request is complete.

At step **730**, the cartridge execution engine **228** sends a message to the transaction manager **606** requesting it to suspend the transaction. At step **732**, the transaction manager **606** sends a suspend request to database servers **608** and **612** to cause them to suspend execution of the transaction. The suspend request includes the globally unique transaction ID so that the database servers **608** and **612** know which transaction to suspend. By sending a suspend request to database servers **608** and **612**, it allows other browsers to execute transactions that are associated with databases **610** and **614**.

At step **734**, transaction manager **606** sends the globally unique transaction ID to the cartridge execution engine **228**. At step **736**, the cartridge execution engine **228** determines whether the sending browser allows for cookie information to be associated with the browser. If the browser does not allow for cookie information to be associated with the browser, at step **738**, the dispatcher **214** is notified that the processing of the revised browser request is complete. Control then returns to step **702** to intercept another revised browser message.

If the browser does allow for cookie information to be associated with the browser, at step **740**, the cartridge execution engine **228** uses the globally unique transaction ID to create cookie information to be associated with the sending browser.

At step **742**, cartridge execution engine **228** forwards the cookie information to dispatcher **214** so that it may be transmitted to the sending browser and notifies the dispatcher **214** that the processing of the revised browser request is complete. Control then returns to step **702** to intercept another revised browser message.

At step **744**, the cartridge execution engine **228** determines whether the revised browser message is associated with a commit transaction URI. In determining whether the revised browser message is associated with a commit transaction URI, the cartridge execution engine **228** uses the cartridge name to identify the previously stored metadata for the type of transaction associated with the revised browser message. Using the previously stored metadata, the cartridge execution engine **228** determines if the revised browser message is associated with a commit transaction URI.

For example, it shall be assumed that the revised browser message contained a cartridge name of STOREFRONT and a URI of /storefront/commit_session. Using the STOREFRONT cartridge name, the cartridge execution engine **228** determines that the revised browser message is associated with the metadata for transaction TX1. Using this metadata, the cartridge execution engine **228** determines that the URI of /storefront/commit_session is associated with a commit transaction.

If the cartridge execution engine **228** determines that the revised browser message is not associated with a commit transaction, then control proceeds to step **774**.

If the cartridge execution engine **228** determines that the revised browser message is associated with a commit transaction, then at step **746**, the cartridge execution engine **228** determines whether the header section of the revised browser message contains cookie information. If cartridge execution engine **228** determines that the header section of the revised browser message contains cookie information, then at step **748** the cartridge execution engine **228** extracts the globally unique transaction ID from the cookie information. Control then proceeds to **752**.

If cartridge execution engine **228** determines that the header section of the revised browser message does not contain cookie information, then at step **750** the cartridge execution engine **228** extracts the globally unique transaction ID from the annotated URI.

At step **752**, the cartridge execution engine **228** packages a resume transaction identifier (tx_resume) into a transaction control message. The cartridge execution engine **228** then sends the transaction control message to the transaction manager **606**.

At step **754**, upon receiving the resume transaction identifier, the transaction manager **606** sends a resume request to database servers **608** and **612** to cause them to resume execution of the transaction. The resume request includes the globally unique transaction ID which allows the database servers **608** and **612** to identify the multiple-request transaction that is associated with the current transaction request.

At step **756**, the cartridge execution engine **228** sends an operation message to cartridge **230** that is based on the

ORCL01622493

transaction information contained in the revised browser message. The operation message also contains a dispatcher pointer that identifies the dispatcher that sent the revised browser request (dispatcher **214**) and allows the cartridge **230** to write information back to the dispatcher. At step **758**, upon receiving the operation message, the cartridge **230** sends a message to the transaction manager **606** requesting handles for access to the databases that are associated with the transaction.

At step **760**, transaction manager **606** returns handles to the appropriate database servers to allow the cartridge **230** to process the transaction request. For example, assuming database **610** is associated with the SEARS database and database **614** is associated with the BANK1 database, transaction manager **606** will return handles to database server **608** and **612** respectively.

At step **762**, cartridge **230** uses the handles returned from transaction manager **606** to execute the operation specified by the operation message that was sent by the cartridge execution engine **228**.

At step **764**, the cartridge **230** determines whether the sending browser allows cookie information to be associated with the browser. If the browser does not allow for cookie information to be associated with the browser, at step **766**, the cartridge **230** causes the globally unique transaction ID to be removed from the annotated hyperlinks of any HTML page that is associated with the transaction. By removing the transaction ID annotations from the hyperlinks of the HTML page, subsequent browser requests that are issued in response to selection of a hyperlink from the HTML page will not contain the globally unique transaction ID and, therefore, will not be mistakenly associated with this multiple-request transaction.

At step **768**, the cartridge **230** uses the dispatcher pointer to return the HTML page generated in response to performing the operation specified in the browser request to the dispatcher **214** and notifies cartridge execution engine **228** that execution of the transaction request is complete.

At step **770**, the cartridge execution engine **228** sends a transaction control message to the transaction manager **606** requesting it to commit the transaction. At step **771**, the transaction manager **606** sends a commit request to database servers **608** and **612** to cause all changes made in response to the various browser requests that belonged to the multiple-request transaction to be committed as an atomic unit of work. The commit request includes the globally unique transaction ID which allows the database servers **608** and **612** to identify the associated multiple-request transaction.

At step **772**, the cartridge execution engine **228** notifies the dispatcher **214** that the processing of the revised browser request is complete and signals the dispatcher **214** to cause the cookie information associated with the committed multiple-request transaction to be removed from the sending browser. By removing the transaction ID from the cookie information associated with the sending browser, subsequent browser requests will not contain the globally unique transaction ID and, therefore, will not be mistakenly associated with the committed multiple-request transaction. Control then returns to step **702** to intercept another revised browser message.

At step **774**, the cartridge execution engine **228** determines whether the revised browser message is associated with a rollback transaction URI. In determining whether the revised browser message is associated with a rollback transaction URI, the cartridge execution engine **228** uses the cartridge name to identify the previously stored metadata that corresponds to the transaction type indicated in the revised browser message. Using the previously stored metadata, the cartridge execution engine **228** determines if the revised browser message contains a rollback transaction URI.

For example, it shall be assumed that the revised browser message contained a cartridge name of STOREFRONT and a URI of /storefront/rollback_session. Using the STOREFRONT cartridge name, the cartridge execution engine **228** determines that the revised browser message is associated with the metadata for transaction TX1 . Using this metadata, the cartridge execution engine **228** determines that the URI of /storefront/rollback_session is associated with a rollback transaction.

If the cartridge execution engine **228** determines that the revised browser message is not associated with a rollback transaction, then control proceeds to step **804**.

If the cartridge execution engine **228** determines that the revised browser message is associated with a rollback transaction, then at step **776**, the cartridge execution engine **228** determines whether the header section of the revised browser message contains cookie information. If cartridge execution engine **228** determines that the header section of the revised browser message contains cookie information, then at step **778** the cartridge execution engine **228** extracts the globally unique transaction ID from the cookie information. Control then proceeds to **782**.

If cartridge execution engine **228** determines that the header section of the revised browser message does not contain cookie information, then at step **780** the cartridge execution engine **228** extracts the globally unique transaction ID from the annotated URI.

At step **782**, the cartridge execution engine **228** incorporates a resume transaction identifier (tx_resume) in a transaction control message. The cartridge execution engine **228** then sends the transaction control message to the transaction manager **606**.

At step **784**, upon receiving the resume transaction identifier, the transaction manager **606** sends a resume request to database servers **608** and **612** to cause them to resume execution of the transaction. The resume request includes the globally unique transaction ID which allows the database servers **608** and **612** to identify the multiple-request transaction that is associated with the current transaction request.

At step **786**, the cartridge execution engine **228** sends an operation message to cartridge **230** that is based on the transaction information contained in the revised browser message. The operation message also contains a dispatcher pointer that identifies the dispatcher that sent the revised browser request (dispatcher **214**) and allows the cartridge **230** to write information back to the dispatcher. At step **788**, upon receiving the operation message, the cartridge **230** sends a message to the transaction manager **606** requesting handles for access to the databases that are used in the specified type of transaction.

At step **790**, transaction manager **606** returns handles to the appropriate database servers to allow the cartridge **230** to process the transaction request. For example, assuming database **610** is associated with the SEARS database and database **614** is associated with the BANK1 database, transaction manager **606** will return handles to database server **608** and **612** respectively.

At step **792**, cartridge **230** uses the handles returned from transaction manager **606** to execute the transaction informa-

ORCL01622494

tion associated with the operation message that was sent by the cartridge execution engine **228**.

At step **794**, the cartridge **230** determines whether the sending browser allows cookie information to be associated with the browser. If the browser does not allow for cookie information to be associated with the browser, at step **796**, the cartridge **230** causes the globally unique transaction ID to be removed from the annotated hyperlinks of any HTML page to be returned to the browser. By removing the transaction ID annotations from the hyperlinks of the HTML page, subsequent browser requests will not contain the globally unique transaction ID and, therefore, will not be mistakenly associated with this multiple-request transaction.

At step **798**, the cartridge **230** uses the dispatcher pointer to return the HTML page that is associated with executing the transaction back to the dispatcher **214** and notifies cartridge execution engine **228** that execution of the transaction request is complete.

At step **800**, the cartridge execution engine **228** sends a transaction control message to the transaction manager **606** requesting it to rollback the transaction. At step **801**, the transaction manager **606** sends a rollback request to database servers **608** and **612** to cause all changes made in response to the browser requests that belong to the multiple-request transaction to be rolled back as an atomic unit of work. The roll back request includes the globally unique transaction ID which allows the database servers **608** and **612** to identify and roll back the correct multiple-request transaction.

At step **802**, the cartridge execution engine **228** notifies the dispatcher **214** that the processing of the revised browser request is complete and signals the dispatcher **214** to cause the cookie information associated with the rolled back multiple-request transaction to be removed from the sending browser. By removing the transaction ID from the cookie information associated with the sending browser, subsequent browser requests will not contain the globally unique transaction ID and, therefore, will not be mistakenly associated with the rolled back multiple-request transaction. Control then returns to step **702** to intercept another revised browser message.

At step **804**, the cartridge execution engine **228** determines whether the header section of the revised browser message contains cookie information. If cartridge execution engine **228** determines that the header section of the revised browser message contains cookie information, then at step **806** the cartridge execution engine **228** extracts the globally unique transaction ID from the cookie information. Control then proceeds to **810**.

If cartridge execution engine **228** determines that the header section of the revised browser message does not contain cookie information, then at step **808** the cartridge execution engine **228** extracts the globally unique transaction ID from the annotated URI.

At step **810**, the cartridge execution engine **228** packages a resume transaction identifier (tx_resume) in a transaction control message. The cartridge execution engine **228** then sends the transaction control message to the transaction manager **606**.

At step **812**, upon receiving the resume transaction identifier, the transaction manager **606** sends a resume request to database servers **608** and **612** to cause them to resume execution of the transaction. The resume request includes the globally unique transaction ID which allows the database servers **608** and **612** to identify the multiple-request transaction that is associated with the current transaction request.

At step **814**, the cartridge execution engine **228** sends an operation message to cartridge **230** that is based on the transaction information contained in the revised browser message. The operation message also contains a dispatcher pointer that identifies the dispatcher that sent the revised browser request (dispatcher **214**) and allows the cartridge **230** to write information back to the dispatcher. At step **816**, upon receiving the operation message, the cartridge **230** sends a message to the transaction manager **606** requesting handles for access to the databases that are associated with the transaction.

At step **818**, transaction manager **606** returns handles to the appropriate database servers to allow the cartridge **230** to process the transaction request. For example, assuming database **610** is associated with the SEARS database and database **614** is associated with the BANK1database, transaction manager **606** will return handles to database servers **608** and **612** respectively.

At step **820**, cartridge **230** uses the handles returned from transaction manager **606** to execute the operation specified in the operation message that was sent by the cartridge execution engine **228**.

At step **822**, the cartridge **230** determines whether the sending browser allows cookie information to be associated with the browser. If the browser does not allow for cookie information to be associated with the browser, at step **824**, the cartridge **230** causes the hyperlinks of an HTML page generated in response to performing the operation to be annotated to include the globally unique transaction ID. By annotating the hyperlinks of the HTML page, the URIs in subsequent browser requests that are issued in response to selecting the links in the HTML page will contain the globally unique transaction ID.

At step **826**, the cartridge **230** uses the dispatcher pointer to return the HTML page thus generated back to the dispatcher **214** and notifies cartridge execution engine **228** that execution of the transaction request is complete.

At step **828**, the cartridge execution engine **228** sends a message to the transaction manager **606** requesting it to suspend the transaction. At step **830**, the transaction manager **606** sends a suspend request to database servers **608** and **612** to cause them to suspend execution of the transaction. The suspend request includes the globally unique transaction ID which allows the database servers **608** and **612** to accurately identify the multiple-request transaction to be suspended. Control then returns to step **702** to intercept another revised browser message.

TRANSACTION TIMEOUTS

According to one embodiment of the invention, a timeout value is associated with each transaction. The timeout value is used to identify multiple-request transactions that have not been active for a specified time period. In one embodiment, each database server maintains a timeout value for the multiple-request transactions that are being serviced by the database server. Thus, whenever a multiple-request transaction begins to execute, the associated database server initializes the timeout value for the particular transaction. Upon receiving a resume transaction request that is associated with a globally unique transaction ID, the database server resets the timeout value for the multiple-request transaction that is associated with the globally unique transaction ID. If a multiple-request transaction times out, the database server causes all changes made as part of the multiple-request transaction to be rolled back as an atomic unit of work. Once the multiple-request transaction is rolled back, a message is then sent to the associated browser to indicate the state of the transaction.

ORCL01622495

CONDUCTING TRANSACTIONS IN A STATELESS WEB ENVIRONMENT

The present invention provides a practical and highly scalable mechanism for conducting multiple-request transactions in a stateless environment, such as the web. According to the invention, a transaction manager is used to coordinate the overall transaction process. Preferably, the transaction manager coordinates the process in such a way that state information is maintained for a transaction without requiring the transaction manager itself to persistently maintain the state information.

In a preferred embodiment, processing of a client request is performed as follows. The transaction manager receives a request from a client, and if the request is a transaction request, the manager initiates a transaction with a transaction processing mechanism, such as a database management system (DBMS). Once the transaction is initiated, the manager preferably forwards the request to another entity, such as an application, which actually processes the request. After the request is processed, control is returned to the manager, and at that point, the manager assembles a set of state information associated with the transaction. This state information may include the identity of the client, the ID and status of the transaction, and what has already transpired in the transaction. Once assembled, the state information, along with the response to the client request, is sent back to the client to be maintained by the client. The state information may be sent to the client in the form of a "cookie" or it may be incorporated into a URL that is returned to the client. While it is possible to do so, state information is preferably not persistently maintained by the manager or by the application that processed the request.

When the client submits a second request relating to the same transaction, the client sends along the state information previously provided by the manager. Upon receiving the second request, the manager extracts the state information from the request, and uses it to resume the previously initiated transaction with the DBMS. Once the transaction is resumed, the manager sends the second request, including the state information, to another entity (the same or a different application) for processing. After the second request is processed, the manager updates the state information associated with the transaction, and sends the updated state information, along with the response to the second request, to the client. The client will send this updated state information in a future request to resume the transaction. This process repeats until the transaction is either committed or rolled back.

The present invention provides several significant advantages. First, note that the transaction manager and the applications that process the requests remain stateless. That is, the transaction manager and the applications are not required to maintain any of the state information for the transaction. All of that information is maintained by the client. This means that no overhead is incurred for storing the information. More importantly, the fact that the client maintains its own state information means that any request from the client can be processed by any thread, process, or node. This significantly improves scalability because it eliminates the need to have a dedicated process or thread for each client.

Another point to note is that even though the client is maintaining the state information, the client is not aware that it is maintaining transaction-specific state information. As discussed above, the state information is provided to the client by the transaction manager. The client simply sends this information back to the transaction manager when it makes its next request. The client is not, nor does it need to be, aware that it is maintaining state information. This is a very advantageous aspect of the present invention because it obviates the need to put any state management logic on the client. This in turn means that no changes or additions need to be made to the client for the present invention to operate properly.

Hence, the present invention provides a practical, scalable, and effective mechanism for conducting transactions in a stateless environment. These and other advantages of the invention will become apparent as the invention is described in further detail.

INCORPORATION OF STATE INFORMATION IN URLS

The present invention provides an effective and highly scalable mechanism for supporting multiple-request operations (including but not limited to transactions) in a stateless environment, such as the web. According to the invention, a server is preferably used to coordinate the overall processing of client requests. Preferably, the server performs this coordination function in such a way that: (1) state information associated with multiple-request operations is maintained by the clients making the requests; (2) the clients are unaware that they are maintaining operation-specific state information; and (3) the server itself is not required to persistently maintain the state information, thereby remaining stateless.

In a preferred embodiment, processing of a client request is performed as follows. The server receives a request from a client, and if the request is for a multiple-request operation, the server initiates an operation. Once the operation is initiated, the server may either forward the request to another entity (such as an application) for processing, or the server may process the request itself. After the request is processed, the server assembles a set of state information associated with the operation. This state information may include the identity of the client, the ID and status of the operation, what has already transpired in the operation, and any other context information associated with the operation. Once assembled, the state information is incorporated into a URL. This URL, along with the response to the client request, is sent back to the client to be maintained by the client. This state information is preferably not persistently maintained by the server.

When the client submits a second request relating to the same operation, the client sends the URL that was previously provided by the server which contains the state information. Upon receiving the second request, the server extracts the state information from the URL, and uses it to resume the previously initiated operation. With the benefit of this state information, the server can resume the operation at the exact point at which the previous request stopped. Once the operation is resumed, the server either processes the request, or forwards it to another entity for processing. After the second request is processed, the server updates the state information associated with the operation, and incorporates the updated state information into another URL. This URL, along with the response to the second request, is sent back to the client to be maintained by the client. The client will send this URL in a future request to resume the operation. This process repeats until the operation is either completed or canceled.

The present invention provides several significant advantages. First, note that the server remains stateless. That is, the server is not required to maintain any of the state informa-

ORCL01622496

tion for the transaction. All of that information is maintained by the client. This means that no overhead is incurred for storing the information. More importantly, the fact that the client maintains its own state information means that any request from the client can be processed by any thread, process, or node. This significantly improves scalability because it eliminates the need to have a dedicated process or thread for each client.

Another point to note is that even though the client is maintaining the state information, the client is not aware that it is maintaining operation-specific state information. As discussed above, the state information is provided by the server to the client in the form of a URL. The client simply sends this URL whenever it requests service from the server. The client treats this URL like any other URL. The client is not, nor does it need to be, aware that this URL contains state information. This is a very advantageous aspect of the present invention because it obviates the need to put any state management logic on the client. This in turn means that no changes or additions need to be made to the client for the present invention to operate properly.

Hence, the present invention provides a practical, scalable, and effective mechanism for supporting multiple-request operations in a stateless environment. These and other advantages of the invention will become apparent as the invention is described in further detail.

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 a restrictive sense.

What is claimed is:

1. A method for processing multiple-request transactions in a stateless environment, wherein the multiple-request transactions involve operations specified in browser messages, the method comprising the steps of:

a cartridge execution engine intercepting browser messages directed to a cartridge; said cartridge execution engine determining whether said browser messages are associated with transactions:

if said browser messages are associated with transactions, then

said cartridge execution engine sending transaction control messages that are based on said browser messages to a transaction manager that is implemented separately from said cartridge;

said cartridge execution engine sending operation messages that are based on said browser messages to said cartridge

in response to said operation messages from said cartridge execution engine, said cartridge performing the operations specified in said operation messages without the cartridge persistently maintaining state information for the multiple-request transactions to which the operations belong; and

in response to said transaction control messages from said cartridge execution engine, said transaction manager causing the operations specified in said operation messages that are performed by said cartridge as part of the multiple-request transactions to be either committed or rolled back as an atomic unit of work.

2 The method of claim 1, wherein the step of causing the operations specified in said operation messages to be committed includes the step of said transaction manager sending commit messages to one or more database servers, wherein the commit messages cause said one or more database servers to commit changes associated with said multiple-reauest transactions as an atomic unit of work.

3. The method of claim 1, wherein the step of causing the operations specified in said operation messages to be rolled back includes the step of said transaction manager sending rollback messages to one or more database servers, wherein the rollback messages cause said one or more database servers to roll back changes associated with said multiple-recst transactions as an atomic unit of work.

4. The method of claim 1, wherein the browser messages associated with transactions are associated with transaction IDs, wherein the transaction IDs identify a browser associated with a particular browser message.

5. The method of claim 4, wherein the transaction IDs are maintained as cookies, wherein the cookies are maintained on the browser that is associated with the particular browser message.

6. The method of claim 4, wherein the transaction IDs are maintained as URLs that are associated with one or more tags in one or more Web pages that are displayed at the browser that is associated with the particular browser message.

7. The method of claim 1, wherein the step of said cartridge execution engine determining whether said browser messages are associated with transactions includes the steps of:

obtaining a URL that is associated with a particular browser message; and

using the URL associated with the particular browser message to determine the state of a transaction that is associatd with the particular browser message.

8. The method of claim 4, wherein the transaction IDs are associated with a timeout period, wherein the expiration of the timeout period indicates that the transaction associated with the transaction ID should be deemed invalid.

9. The method of claim 1, wherein:

prior to intercepting browser messages directed to the cartridge,

registering the cartridge, wherein the cartridge is registered by storing metadata that defines a set of attributes that is associated with one or more transaction types.

10. The method of claim 1, wherein the step of said cartridge execution engine determining whether said browser messages are associated with transactions includes the steps of:

retrieving metadata based on the intercepted browser messages; and

using the retrieved metadata to determine whether the browser messages are associated with transactions.

11. A computer readable medium carrying sequences of instructions for processing multiple-request transactions in a stateless environment, wherein the multiple-request transactions involve operations specified in browser messages, the sequences of instructions including instructions for performing the steps of:

a cartridge execution engine intercepting browser messages directed to a cartridge;

said cartridge execution engine determining whether said browser messages are associated with transactions;

if said browser messages are associated with transactions, then

said cartridge execution engine sending transaction control messages that are based on said browser

ORCL01622497

messages to a transaction manager that is implemented separately from said cartride;

said cartridge execution engine sending operation messages that are based on said browser messages to said cartridge

in response to said operation messages from said cartridge execution engine, said cartridge performing the operations specified in said operation messages without the cartridge persistently maintaining state information for the multiple-request transactions to which the operations belong; and

in response to said transaction control messages from said cartridge execution engine, said transaction manager causing the operations specified in said operation messages that are performed by said cartridge as part of the multiple-request transactions to be either committed or rolled back as an atomic unit of work.

12. The computer readable medium of claim **11**, wherein the browser messages associated with transactions are associated with transaction IDs, wherein the transaction IDs identify a browser associated with a particular browser message.

13. The computer readable medium of claim **11**, wherein the step of said cartridge execution engine determining whether said browser messages are associated with transactions includes the steps of:

obtaining a URL that is associated with a particular browser message; and

using the URL associated with the particular browser message to determine the state of a transaction that is associate with the particular browser message.

14. A system for processing multiple-request transactions in a stateless environment, wherein the multiple-request transactions involve operations specified in browser messages, the system comprising:

a memory;

one or more processors coupled to the memory; and

a set of computer instructions contained in the memory, the set of computer instructions including computer instructions which when executed by the one or more processors, cause the one or more processors to perform the steps of:

a cartridge execution engine intercepting browser messages directed to a cartridge;

said cartridge execution engine determining whether said browser messages are associated with transactions;

if said browser messages are associated with transactions, then

said cartridge execution engine sending transaction control messages that are based on said browser messages to a transaction manager that is implemented separately from said cartridge;

said cartridge execution engine sending operation messages that are based on said browser messaes to said cartridge;

in response to said operation mnessaes from said cartridge execution engine, said cartridge perfoming the operations specified in said operation messages without the cartridge persistently maintaining state information for the multiple-request transactions to which the operations belong; and

in response to said transaction control messages from said cartridge execution engine, said transaction manager causing the operations specified in said operation messages that are performed by said cartridge as part of the multiple-request transactions to be either committed or rolled back as an atomic unit of work.

15. The system of claim **14**, wherein the browser messages associated with transactions are associated with transaction IDs, wherein the transaction IDs identify a browser associated with a particular browser message.

16. The system of claim **14**, wherein the step of said cartridge execution engine determining whether said browser messages are associated with transactions includes the steps of:

obtaining a URL that is associated with a particular browser message; and

using the URL associated with the particular browser message to determine the state of a transaction that is associate with the particular browser message.

17. The method of claim **1**, wherein:

the step of said cartridge execution engine intercepting browser messages includes the step of said cartridge execution engine intercepting browser messages that include a begin transaction command; and

in response to said cartridge execution engine receiving a browser message that includes a begin transaction command, said cartridge execution engine sending a transaction control message to said transaction manager to cause said transaction manager to begin said transaction.

18. The method of claim **1**, wherein:

the step of said cartridge execution engine intercepting browser messages includes the step of said cartridge execution engine intercepting browser messages that include a commit transaction command; and

in response to said cartridge execution engine receiving a browser message that includes a commit transaction command, said cartridge execution engine sending a transaction control message to said transaction manager to cause said transaction manager to commit said transaction.

19. The method of claim **1**, wherein:

the step of said cartridge execution engine intercepting browser messages includes the step of said cartridge execution engine intercepting browser messages that include a rollback transaction command; and

in response to said cartridge execution engine receiving a browser message that includes a rollback transaction command, said cartridge execution engine sending a transaction control message to said transaction manager to cause said transaction manager to roll back said transaction.

20. The method of claim **17**, further comprising the step of receiving said begin transaction command in the form of a URL at said cartridge execution engine in response to selection of a control associated with a tag of a Web page displayed at the browser.

21. The method of claim **18**, further comprising the step of receiving said commit transaction command in the form of a URL at said cartridge execution engine in response to selection of a control associated with a tag of a Web page displayed at the browser.

22. The method of claim **19**, further comprising the step of receiving said rollback transaction command in the form of a URL at said cartridge execution engine in response to selection of a control associated with a tag of a Web page displayed at the browser.

23. The computer readable medium of claim **11**, wherein the step of causing the operations specified in said operation

ORCL01622498

messages to be committed includes the step of said transaction manager sending commit messages to one or more database servers, wherein the commit messages cause said one or more database servers to commit changes associated with said multiple-request transactions as an atomic unit of work.

24. The computer readable medium of claim **11**, wherein the step of causing the operations specified in said operation messages to be rolled back includes the step of said transaction manager sending rollback messages to one or more database servers, wherein the rollback messages cause said one or more database servers to roll back changes associated with said multiple-request transactions as an atomic unit of work.

25. The computer readable medium of claim **11**, wherein

the step of said cartridge execution engine intercepting browser messages includes the step of said cartridge execution engine intercepting browse- messages that include a begin transaction command; and

the computer readable medium further comprising instructions for performing the step of, in response to said cartridge execution engine receiving a browser message that includes a begin transaction command, said cartridge execution engine sending a transaction control message to said transaction manager to cause said transaction manager to begin said transaction.

26. The computer readable medium of claim **11**, wherein

the step of said cartridge execution engine intercepting browser messages includes the step of said cartridge execution engine intercepting browser messages that include a commit transaction command; and

the computer readable medium further comprising instructions for performing the step of, in response to said cartridge execution engine receiving a browser message that includes a commit transaction command, said cartridge execution engine sending a transaction control message to said transaction manager to cause said transaction manager to commit said transaction.

27. The computer readable medium of claim **11**, wherein

the step of said cartridge execution engine intercepting browser messages includes the step of said cartridge execution engine intercepting browser messages that include a rollback transaction command; and

the computer readable medium further comprising instructions for performing the step of, in response to said cartridge execution engine receiving a browser message that includes a rollback transaction command, said cartridge execution engine sending a transaction control message to said transaction manager to cause said transaction manager to roll back said transaction.

28. The computer readable medium of claim **12**, further comprising instructions for maintaining the transaction IDs as cookies, wherein the cookies are maintained on the browser that is associated with the particular browser message.

29. The computer readable medium of claim **12**, further comprising instructions for maintaining the transaction IDs as URLs, wherein the URLs are associated with one or more tags in one or more Web pages that are displayed at the browser that is associated with the particular browser message.

30. The computer readable medium of claim **12**, further comprising instructions for associating a timeout period with the transaction IDs, wherein the expiration of the timeout period indicates that the transaction associated with the transaction ID should be deemed invalid.

31. The computer readable medium of claim **11**, further comprising instructions for performing the steps of prior to intercepting browser messages directed to the cartridge,

registering the cartridge, wherein the cartridge is registered by storing metadata that defines a set of attributes that is associated with one or more transaction types.

32. The computer readable medium of claim **11**, wherein the step of said cartridge execution engine determining whether said browser messages are associated with transactions includes the steps of:

retrieving metadata based on the intercepted browser messages; and

using the retrieved metadata to determine whether the browser messages are associated with transactions.

33. The computer readable medium of claim **25**, further comprising instructions for performing the step of receiving said begin transaction command in the form of a URL at said cartridge execution engine in response to selection of a control associated with a tag of a Web page displayed at the browser.

34. The computer readable medium of claim **26**, further comprising instructions for performing the step of receiving said commit transaction command in the form of a URL at said cartridge execution engine in response to selection of a control associated with a tag of a Web page displayed at the browser.

35. The computer readable medium of claim **27**, further comprising instructions for performing the step of receiving said rollback transaction command in the form of a URL at said cartridge execution engine in response to selection of a control associated with a tag of a Web page displayed at the browser.

36. The system of claim **14**, wherein the step of causing the operations specified in said operation messages to be committed includes the step of said transaction manager sending commit messages to one or more database servers, wherein the commit messages cause said one or more database servers to commit changes associated with said multiple-request transactions as an atomic unit of work

37. The system of claim **14**, wherein the step of causing operations specified in said operation messages to be rolled back includes the step of said transaction manager sending rollback messages to one or more database servers, wherein the rollback messages cause said one or more database servers to roll back changes associated with said multiple-request transactions as an atomic unit of work.

38. The system of claim **14**, wherein

the step of said cartridge execution engine intercepting browser messages includes the step of said cartridge execution engine intercepting browser messages that include a begin transaction command; and

in response to said cartridge execution engine receiving a browser message that includes a begin transaction command, said cartridge execution engine sending a transaction control message to said transaction manager to cause said transaction manager to begin said transaction.

39. The system of claim **14**, wherein

the step of said cartridge execution engine intercepting browser messages includes the step of said cartridge execution engine intercepting browser messages that include a commit transaction command; and

in response to said cartridge execution engine receiving a browser message that includes a commit transaction command, said cartridge execution engine sending a transaction control message to said transaction manager to cause said transaction manager to commit said transaction.

ORCL01622499

generating software that generates a dynamic Web page, as described by the Specification. In one embodiment, Page servers reside on a separate machine to accomplish the claimed function. (Lowery: Col. 5 line 49-50.)

Claim 9:

[a] A networked system for managing a dynamic Web page generating request, said system comprising:

The Derler Thesis discloses a networked system for managing dynamic Web page generation requests from a WWW client sent to a Web server (WWW Gateway). Introduction, pp. 1-2; § 3.3.4 at p. 37; § 4.1 at pp. 43-44; § 4.7.2 at pp. 94-96; Figs. 4.16-4.17 at p. 95 (depicting networked system of WWW client, WWW Gateway, Hyper-G Server and remote servers (page servers) and document cache (page cache)).

[b] one or more data sources;

Derler: See id; Figure 4.17 (Hyper-G server, remote server and document cache are data sources.)

[c] a page server having a processing means;

Derler: "processing means" refers to a general-purpose processor or microprocessor 102 (Figure 1) and equivalents thereof, col. 3 lines 25 et seq. § 4.4.2, "The class WWWSlave contains all data and code necessary to process requests (slave process as a page server) from a single WWW client..." Alternately, The Hyper-G server is a page server (serves pages to Slave Process) and as such inherently has a general-purpose processor, the same as or equivalent to the processing means of the '554 patent. Derler, § 3.1 at p. 32; Figs. 4.16-4.17 and accompanying disclosure at pp. 94-95.

[d] a first computer system including means for generating said request; and

Derler: "means for generating said request" refers to a Web client in the form of a Web browser and equivalents thereof. Col. 4 lines 11-17. See Derler, § 1 at p. 1; § 3.3.4 at p. 37; Figs. 4.16-4.17 and accompanying disclosure at pp. 94-95 (depicting WWW Client, which is a first computer system which has a browser generating a Web page request). See also Derler at p. 4 ("WWW clients, often called 'WWW browsers'"); p. 48; pp. 67-68 (explaining user interaction with Hyper-G systems with "WWW clients like Mosaic or Netscape", both of which are Web browsers).

[el] a second computer system including

Derler: WWW Gateway is the second computing system. See generally Chapter 4.

[e2] means for receiving said request from said first computer,

Derler: The means for receiving request is Web server executable 201E (Figure 4) and equivalents thereof. Derler's Master process is a Web server executable process, same as or equivalent to '554 patent Web server executable. See Derler § 4.2.2 at p. 48-49; § 4.2.3 at p. 49-50; Figs. 4.1, 4.16-4.17 and accompanying disclosure at pp. 94-95 (depicting WWW Gateway and Master Process receives a Web page request from WWW client (first computer)).

[e3] said second computer system also including a router, said router routing said request from said second computer system to said page server,

Derler: See Id; see also Fig. 4.2, p. 52, route to passing process, p. 49-50, (alternately) Child/Slave processes are a "router" within meaning of the '554 patent since they route request to Hyper-G page server.

[e4] wherein said routing further includes intercepting said request at said second computer,

Derler: § 4.2.2 at pp. 48-49; § 4.2.3 at pp. 49-50; Figs. 4.16-4.17 and accompanying disclosure at pp. 94-95; forking process of Master indicates interception of WWW request by "Passing Process".

[e5] routing said request from said second computer to a dispatcher, and

Request is routed from Master Server to Passing Process. Passing Process selects appropriate Slave Process and "dispatches" request to said Slave Process for processing. Derler: § 4.2.3 at pp. 49-50

[e6] dispatching said request to said page server

Derler: See id.; Figure 4.1, page server is Hyper-G server; Passing Process dispatches a Hyper-G request to the Slave Process (page server), which in turn may connect to a Hyper-G server. See also discussion of "dispatching" for element [b6] for claim 1, supra.

[e7] said page server receiving said request and

Derler: See 4.2.3, Slave Process receives request from Passing Process.

[e8] releasing said second computer system to process other requests,

Derler: § 4.2.2 at pp. 48-49; see also, e.g., Fig 4.5 and accompanying disclosure at pp. 49-54 (depiction of WWW Gateway processing multiple concurrent requests).

[e9] said page server processing means processing said request and dynamically generating a Web page in response to said request,

Derler: § 3.3.4 at p. 37; pp. 52-54 and Figs. 4.16-4.17 and accompanying disclosure at pp. 94-95 (the general-purpose processor ("processing means") in the Hyper-G page server and/or remote page servers process request). See also, e.g., § 4.4.2 at p. 65; Fig. 4.10 at p. 71 (sample of a dynamically-generated Web page in response to request). See also § 4.2.3 at p. 50: "When [a slave process (page server) on the WWW gateway] has accepted a connection initiated by the passing process, it reads the request, processes it (usually by transforming into a Hyper-G request for the server), receives the response from the Hyper-G server, transforms the response again if necessary, sends it to the passing process, and closes the connection to the passing process." Inherently, if the client receives a response from the Hyper-G server, the Hyper-G server has processed the request. (Content may be served from the Hyper-G server (page server), received and processed at the Slave process (page server), and in turn received at the Passing Process (page server), prior to being delivered as a request response to the client. Each of these processes may be involved in "serving pages".)

[e10] said Web page including data dynamically retrieved from said one or more data sources.

Derler: § 3.3.4 at p. 37; Fig. 4.10 at p. 71 (depicting sample dynamically-generated Web pages from multiple sources); data source can be data cache, see § 3.3.5; see p. 95 Figure 4.17, data source can be a remote server.

Claim 10:

[a] The networked system in claim 9 wherein said router in said second computer system includes:

Derler: § 4.2.3 at pp. 49-50 (Master Process, Slave Process, and Passing Process interact. The WWW Gateway "second computer system" involves the Master Process; Figs. 4.16-4.17 and accompanying disclosure at pp. 94-95.

[b] an interceptor intercepting said request at said second computer system and routing said request; and

Derler: § 4.2.2 at p. 48; § 4.2.3 at pp. 49-50 (master and forking process comprise an interceptor / Passing Process) As noted above, the Passing Process "intercepts" the request, and makes a determination as to which Slave Process to route the request.

[c] a dispatcher receiving said routed request from said interceptor and dispatching said request to said page server.

The Passing Process dispatches the request to the appropriate Slave Process (page server). The Slave Process and the Hyper-G Server work together to serve pages. Derler: § 4.2.3 at pp. 49-50; Figs. 4.1-4.4

Claim 11:

[a] A machine readable medium having stored thereon data representing sequences of instructions, which when executed by a computer system, cause said computer system to perform the steps of:

Derler: The Hyper-G system is a computer system executing software instructions which are inherently stored in a machine readable medium. § 1 at p. 1; § 3.3.4 at p. 37; § 3.1 at p. 32. See also Fig. 4.16-4.17 at p. 95; § 4.6.3. See also generally Chapter 4 (WWW gateway is a computer system which includes software).

[bl] routing a dynamic Web page generation request from a Web server to a page server,

Derler: § 4.2.2 at p. 49; § 4.2.3 at pp. 50, 52, § 4.7.2 at p. 95; Figs. 4.16-17; Web page requests are dynamically routed from WWW Gateway ("Web server" / Master Process) to page server (Slave Process and Hyper- G server or a remote server work together to serve pages).

[b2] said page server receiving said request and

Derler: The client Web page request is received by the page server for processing. See § 4.2.3.

[b3] releasing said Web server to process other requests

Derler: Derler's WWW Gateway uses parallel processing of Web page requests wherein the WWW Gateway is released from one request to process other requests. See § 4.2.2 at pp. 48-49; § 4.2.3 at p. 49 (Master process of WWW Gateway), Figs. 4.1, 4.2, 4.3, 4.4 and 4.5 (pp. 52-55) illustrating process of releasing WWW Gateway to process other requests in parallel.

[b4] wherein said routing step further includes the steps of intercepting said request at said Web server,

Derler: § 4.2.2 at pp. 48-49; § 4.2.3 at p. 49; Figs. 4.16-4.17 and accompanying disclosure at pp. 94-95; forking process of Master indicates interception of WWW request by Passing Process.

[b5] routing said request from said Web server to a dispatcher, and

Derler: § 4.2.3 at pp. 49-50, Web Server routes request to Passing Process. Passing Process dispatches request to appropriate Slave Process. Fig. 4.7

[b6] dispatching said request to said page server;

Derler: See Id; Passing Process dispatches request to appropriate Slave Process (page server), which in turn may dispatch request to Hyper-G server (alternate page server).

[c] processing said request, said processing being performed by said page server while said Web server concurrently processes said other requests; and

Derler: § 4.2.2 at pp. 48-49; § 4.2.3 at p. 49, pp. 52-54, processing of request by Slave Process (page server); Fig. 4.7, such processing occurs concurrently with the WWW Gateway processing other requests. See also § 4.2.3 at p. 50 ("When it [a Slave process on the WWW Gateway] has accepted a connection initiated by the passing process, it reads the request, processes it (usually by transforming it into a Hyper-G request for the server, receives the response from the Hyper-G server, transforms the response again if necessary, sends it to the passing service, closes the connection to the passing process.") The client receives a response from Passing Process (Fig.4.7), after the Slave Process (page server) and possibly with further processing by the Hyper-G server (alternate page server), has processed the request.

"After accepting the connection, the Master forks generating a child process. Immediately after that, the master closes its connection to W3client A and is then ready to accept other connections (Web server concurrently processes said other requests)."

[d] dynamically generating a Web page, said Web page including data retrieved from one or more data sources.

Derler: § 3.3.4 at p. 37; Fig. 4.10 at p. 71 (depicting sample dynamically-generated Web page from data from one or more sources); data cache and remote server are data sources, § 3.3.5 and p. 95.

Claims 1-4 and 7-11 are rejected under 35 U.S.C. 102(a) as being anticipated by Dienst:

Claim 1:

[a] A computer-implemented method for managing a dynamic Web page generation request to a Web server, said computer-implemented method comprising the steps of:

Lagoze describes a Web-based architecture, called "Dienst," that provides access to distributed document libraries over the World WideWeb (Web / WWW). (See, Abstract at p. 1.) The architecture comprises a Web server, a CGI stub, a Dienst server and a document database (§ 4.1 at p. 9). Using a Web browser, a user can submit a search request for certain documents meeting the search parameters. A so-called "Dienst request" is packaged within the path portion of a standard HTTP request and transmitted from the user's Web client (browser) to the Web server. (§4.1 at p. 11). "Search results are presented as a hypertext document," i.e., a Web page, which is transmitted back to the client as a standard HTTP response. (§ 1 at p. 6; § 4.1 at p. 11.) Because of the dynamic nature of the results, each search request constitutes a "dynamic Web page generation request." The Dienst Manual describes a computer-implemented method for handling dynamic Web page requests. (P. 6; pp. 9-11, Fig. 2.) Since the Web clients accessing the Dienst system use "popular browsers of the World Wide Web [such] as Mosaic, Cello and Netscape" (p. 6), the information presented to the Web clients constitutes "Web pages."

[bl] routing said request from said Web server to a page server,

Dienst Manual: (Pp. 10-11; Fig. 2); Web server is WWW server; page server is Dienst Server. Arrows in Figure 2 illustrate routing of request from WWW server to Dienst Server. (§ 4.1 at p. 10) As described in the Lagoze reference, when the Web server detects a Dienst request in the path portion of an HTTP request received from a Web client, the Web server invokes (routes request through CGI Stub) the CG1 stub. The CGI stub "is a small, simple executable that strips the Dienst request from the HTTP request, packages a few important environment variables that contain information about the request .... and sends (further routing said request) them via socket I/O to the Dienst server." (§ 4.1 at pp. 11.) The Dienst server is a "page server" in that it processes the Dienst request, formulates the results of a search, and transmits a dynamic Web page containing the search results back to the Web client. (Id.)

[b2] said page server receiving said request and

Dienst Manual: The Dienst Server receives the request. (See Id.)

[b3] releasing said Web server to process other requests,

Dienst Manual: (P. 11.) CGI stub strips Dienst request from HTTP request; the CGI stub is invoked each time the gateway Web server receives a Dienst request. Inherently, the WWW server is released from processing that request, and can process other requests. This is further apparent from nature of the Dienst system to provide for simultaneous access to documents on

Dienst system for distributed WWW clients. P. 11, The Web server serves as a front-end for Dienst. The Web server is set up as a gateway to a Dienst server.

[b4] wherein said routing step further includes the steps of intercepting said request at said Web server,

Dienst Manual: As described above [b1], the Web server recognizes from the path portion of the HTTP request that the request is a Dienst request (i.e., a dynamic Web page generation request) and passes the request to the CGI stub, thereby "intercepting" it. That is, the Web server stops or interrupts its processing of the Dienst request and deflects processing of the request to the CGI stub. (P. 11)

[b5] routing said request from said Web server to a dispatcher, and

Dienst Manual: Request is passed to CGI stub (routing process) (Fig. 2; pp. 10-11), dispatching is a function of Dienst CGI stub. The CGI stub serves as a dispatcher that dispatches the request to one or more Dienst servers. (§ 4.1 and 4.2 at pp. 11-12.)

[b6] dispatching said request to said page server;

Dienst Manual: (P. 11); Dienst CGI stub sends (dispatches) request via I/0 socket to Dienst Server (page server). (P. 12), The Dienst Manual describes a Dienst Server which functions as a

user interface server which "provides a front-end for searching the collection provided by all inter-operating Dienst servers." See Figure 2, page 10 "Dienst Server" and "Other Dienst Servers". The "other Dienst servers" host a distributed collection of documents. (P. 12), "Each server is able to locate the indexing and repository site for a specific publisher using tables that are maintained by a distinguished server... The respective user interface server then dispatches search requests, in parallel to the Dienst index servers that corresponds to the publisher selections(s)..."(Dienst Manual, page 12, ¶¶ l, 2.)

The CGI Stub dispatches to the Dienst Server, which may in turn "dispatch" other Dienst Servers or to document databases. (See Fig. 2, p. 10)

[c] processing said request, said processing being performed by said page server while said Web server concurrently processes said other requests; and

Dienst Manual: Fig. 2; pp. 10-11; function of Dienst Server is to process requests and generate Web pages. Because a "Dienst server is a standalone multi-threaded process" (§ 4.1 at p. 11), it necessarily processes the request while the Web server continues to concurrently process other HTTP requests. P. 11, The Web server serves as a front end for Dienst. Inherently, with the CGI stub program intercepting the Dienst request, the Web server process is released from processing that request, and can process other requests concurrently. This is further apparent from the nature of the Dienst system to provide for simultaneous access to documents on the Dienst system from distributed WWW clients.

[dl] dynamically generating a Web page in response to said request,

Dienst Manual: "Search results are presented as a hypertext document," i.e., a Web page, which is transmitted back to the client as a standard HTTP response. (§ 1 at p. 6); "responses from Dienst are formatted" (§ 4.1 at p. 11); The function of Dienst server is to dynamically generate a Web page in response to request; (see also Figs. 4 and 5, pp. 14-15) (showing dynamically generated pages); (p. 58) (describing information that can be provided on dynamically-generated pages).

[d2] said Web page including data dynamically retrieved from one or more data sources.

Dienst Manual: Pp. 6, 9-12, Fig. 2, data sources are repositories of documents on Dienst Server or in a document database; data is dynamically retrieved at time of processing of request. See also Figs. 4 and 5, pp 9-11 & 58.

Claim 2:

The computer-implemented method in claim 1 herein said step of processing said request includes the step of identifying said one or more data sources from which to retrieve said data.

Dienst Manual: "A user of a Dienst server perceives a single logical collection [of documents], even though the collection is distributed over multiple servers. This is accomplished by

interaction between a set of Dienst servers ...." (§ 4.2 at p. 11.) "Each server, in a set of interoperating Dienst servers, is capable of responding to a request for any document in the distributed collection. The respective server uses the publisher in the docid of the requested document and the mapping tables (processing / identifying one or more data sources) downloaded from the meta server to route the document request to the server that acts as the repository for the publisher. This routing is transparent to the user." (Id. at p. 12.) Pp. 11-12, indexing service and distributed searching function of Dienst Server inherently identifies a data source from which to retrieve data since that is the function of the indexing and searching functions.

Claim 3:

The computer-implemented method in claim 2 wherein said step of dynamically generating said Web page includes the step of dynamically retrieving said data from said one or more data sources.

Dienst Manual: P. 11; Dienst server dynamically retrieves data from data source such as a repository of documents. (Figure 2, Document Database) P. 12, distributed document access feature of Dienst server. See also Figs. 4 and 5 (showing pages with dynamically retrieved data).

Claim 4:

The computer-implemented method in claim 3 wherein said step of processing said request includes the step of said page server maintaining a connection cache to said one or more data sources.

Dienst Manual: As mentioned above, in the Dienst architecture, "[a] Dienst server perceives a single logical collection [of documents], even though the collection is distributed over multiple servers. This is accomplished by interaction between a set of Dienst servers..... " (§ 4.2 at p. 11.) Each document in the document database has an associated "docid" and "each docid ... consists of a publisher identifier and a name." (§ 4.2 at p. 12.), "Each server in a set of interoperating Dienst servers indexes and acts as a repository for documents from one or more publishers, in the current version, all the documents for a publisher must be indexed and reside on the same server. Each server is able to locate the indexing and repository site for a specific publisher using tables that are maintained by a distinguished server, the meta server .... This allows every server to provide the next two functions .... distributed document access - Each server, in a set of interoperating Dienst servers, is capable of responding to a request for any document in the distributed collection. The respective server uses the publisher in the docid of the requested document and the mapping tables downloaded from the meta server to route the document request to the server that acts as the repository for the publisher. This routing is transparent to the user." (Id. at p. 12.) Thus, the meta server serves as a "connection cache" in that it maintains information about which Dienst server in a set of interoperating Dienst servers is the repository (i.e., data source) for a given publisher's documents. (Pp. 9, 11-12); server registration process, and/or indexing service, is maintenance of connection cache to data source.

Claim 7:

The computer-implemented method in claim 3 wherein said page server includes custom HTML extension templates for configuring said Web page.

Dienst Manual: P. 6, "Search results are presented as a hypertext document. (configured Web page)" (Pp. 15, 17, 60 and 61) HTML is used for configuring Web pages, e.g., pages containing data in response to queries of Dienst system; HTML image map templates of Fig. 7 (p. 17) are custom HTML extension templates. (See also Figures 4 and 5, pp. 14, 15.)

The Dienst server provides a Web designer who is managing the document database with a variety of document format templates that can be used to provide different formats for display of documents retrieved from the database. These templates include "inline - a paged format that is an image displayed inline on HTML pages" as well as "composite - a paged format that is a composite of thumbnail images." (§ 7 at pp. 33-35; § 5 at p. 17.) With the composite format, "[each] thumbnail page is implemented using a HTML image map, allowing the user to select on a page and dispatch a URL to view that specific page in readable form." § 5 at p. 17.

Claim 8:

The computer-implemented method in claim 7 wherein said step of processing said request further includes the step of inserting said dynamically retrieved data from said one or more data sources into said custom HTML extension templates.

Dienst Manual: P. 17, Fig. 7, disclosing HTML image map templates which are custom HTML extension templates; data retrieved from data sources are inserted into HTML image map templates. See also Figs. 4 and 5, which appear to be based on HTML templates.
When document data is retrieved from the document database it is inserted into one of the document format templates supported by the Dienst server for display to the user. (§ 7 at pp. 33-35; § 5 at p. 17.)

Claim 9:

[a] A networked system for managing a dynamic Web page generation request, said system comprising:

Dienst Manual: (Abstract; pp. 5-6, 9-11); Fig. 2 -showing distributed networked system for managing dynamic Web page generation requests). A Web-based architecture, called "Dienst," that provides access to distributed document libraries over the World Wide Web (Web). (See, Abstract at p. 1.) The architecture comprises a Web server, a CGI stub, a Dienst server and a document database. (§ 4.1 at p. 9.) Using a Web browser, a user can submit a search request for certain documents meeting the search parameters. A so-called "Dienst request" is packaged within the path portion of a standard HTTP request and transmitted from the user's Web client to the Web server. (Lagoze, §4.1 at p. 11.) "Search results are presented as a hypertext document," i.e., a Web page, which is transmitted back to the client as a standard HTTP response. (§ 1 at p. 6; § 4.1 at p. 11.) The Dienst Manual describes a computer-implemented method for handling

dynamic Web page requests. The Dienst system includes a WWW server which manages dynamic Web page requests from WWW clients, and a page server ("Dienst Server"); system is a computer-implemented method as indicated throughout the document. Because of the dynamic nature of the results, each search request constitutes a "dynamic Web page generation request." The Web clients accessing Dienst system use "popular browsers of the World Wide Web [such] as Mosaic, Cello and Netscape" (p. 6), and the information presented to the Web clients constitutes "Web pages".

See comments above regarding 35 U.S.C. § 112, ¶ 6.

[b] one or more data sources;

Dienst Manual (Pp. 6 and 9-12); Dienst servers storing documents in the Dienst system are data sources, as are document databases. The Dienst architecture includes a "document database" consisting of one or more document repositories (data sources). (§ 4.1 at p. 9.) See Fig. 2, p. 10.

[c] a page server having a processing means;

Dienst Manual: Under § 112, ¶ 6, "processing means" refers to a general-purpose processor or microprocessor 102 ('554: Fig. 1) and equivalents thereof ('554": col. 3 lines 25 et seq.) See Dienst Manual (pp. 6 and 9- 12) and Fig. 2. ; Dienst Servers are computers which inherently

include general-purpose processors or equivalent thereof. The Dienst server is a "page server" and it necessarily includes a processing means.

[d] a first computer system including means for generating said request; and

Dienst Manual: Under § 112, ¶ 6, "means for generating said request" refers to a Web client in the form of a Web browser. ('554: Col. 4 lines 11-17.) See Dienst Manual, WWW client, Fig. 2, p. 6 (mentioning popular Web browsers such as Netscape) and p. 10. Using a Web browser, a user can submit a search request for certain documents meeting the search parameters. The user's Web browser constitutes a "first computer system."

[e1] a second computer system including

Dienst Manual: WWW server; Fig. 2; pages 10 and 11. The Web server of the Dienst architecture constitutes a "second computer system" and it is intended to receive requests from client Web browsers. "A Web server that serves as a front end for Dienst..." (p. 11)

[e2] means for receiving said request from said first computer,

Dienst Manual: Under § 112 ¶ 6, the means for receiving request is Web server executable 201E ('554: Figure 4) and equivalents thereof. Dienst WWW server includes an executable as a means for receiving request from WWW client. Fig. 2, pages 10 and 11 (CERN & NCSA servers), 31

and 32 (describing downloading/ installing NCSA or CERN Web server code (executable)). Executable server code on a processor provides the means for receiving a request.

[e3] said second computer system also including a router, said router routing said request from said second computer system to said page server,

Dienst Manual: (Fig. 2; pp. 10-11); Dienst CGI stub acts as a router routing request from WWW server to Dienst Server. As described above, when the Web server detects a Dienst request in the path portion of an HTTP request received from a Web client, the Web server (second computer) invokes the CG1 stub. The CGI stub "is a small, simple executable that strips the Dienst request from the HTTP request, packages a few important environment variables that contain information about the request .... and sends (routes) them via socket 1/O to the Dienst server (page server)." (§ 4.1 at pp. 11.) The Dienst server is a "page server" in that it processes the Dienst request, formulates the results of a search, and transmits a dynamic Web page containing the search results back to the Web client (Id.).

[e4] wherein said routing further includes intercepting said request at said second computer,

Dienst Manual: As described above, the Web server (second computer) recognizes (intercepts) from the path portion of the HTTP request that the request is a Dienst request (i.e., a dynamic Web page generation request) and passes the request to the CGI stub (intercepting at second computer / Web server).

[e5] routing said request from said second computer to a dispatcher, and

Dienst Manual: Pp. 10-11, routing request from Web server to CGI stub / dispatcher. P. 11; Web server (second computer) recognizes (intercepts) from the path portion of the HTTP request that the request is a Dienst request (i.e., a dynamic Web page generation request) and passes (route from second computer to dispatcher) the request to the CGI stub (intercepting at second computer / Web server) ("the stub is a small, simple executable that strips the Dienst request from the HTTP request."). The CGI stub serves as a dispatcher that dispatches the request to one or more Dienst servers. (§ 4.1 and 4.2 at pp. 11-12.)

[e6] dispatching said request to said page server,

Dienst Manual P. 11; CGI stub send (dispatches) request via I/O socket to Dienst Server (page server). The CGI stub serves as a dispatcher that dispatches the request to one or more Dienst servers. (§ 4.1 and 4.2 at pp. 11-12.) See additional "dispatching" comments at claim 1 [b6], supra.

[e7] said page server receiving said request and

Dienst Manual: (Fig. 2; pp. 10 and 11); Dienst Server receives request.

[e8] releasing said second computer system to process other requests,

Dienst Manual: P. 11. CGI stub strips Dienst request from HTTP request; the CGI stub is invoked each time the gateway Web server (second computer) receives a Dienst request. Inherently, the WWW server (second computer) is released from processing that request, and can process other requests. "A Web server that serves as a front-end for Dienst…", (p. 11).

[e9] said page server processing means processing said request and dynamically generating a Web page in response to said request,

Dienst Manual: Fig. 2; pp. 9-12; function of general-purpose processor ("processing means"), in the Dienst Server is to process requests and generate Web pages. See also Figs. 4 and 5, pp. 14-15 (showing dynamically generated pages); p. 58 (describing information that can be provided on dynamically-generated pages). A Dienst server retrieves information from the document database that meets a user's search criteria and generates a Web page containing the results of the search. (§ 1 at p. 6; § 4.1 at p. 11) ("Search results are presented as a hypertext document," i.e., a Web page, which is transmitted back to the client as a standard HTTP response). Because each search can be different, the process of responding to search requests necessarily is "dynamic."

[e10] said Web page including data dynamically retrieved from said one or more data sources.

Dienst Manual: (Pp. 6, 9-12), Fig. 2, data sources are repositories of documents on Dienst Server or in a document database; data is dynamically retrieved at time of processing of request. (See also Figs. 4 and 5, pp. 14-15) -showing dynamically generated pages); (p. 58)-describing information that can be provided on dynamically generated pages.

Claim 10:

[a] The networked system in claim 9 wherein said router in said second computer system includes:

Dienst Manual: As described above, the Web server (second computer) recognizes from the path portion of the HTTP request that the request is a Dienst request (i.e., a dynamic Web page generation request) and passes (routes) the request to the CGI stub. (Pp. 10-11; Fig. 2); WWW server (second computer) functions as a router to CGI Stub.

[b] an interceptor intercepting said request at said second computer system and routing said request; and

Dienst Manual: (Id.) – Fig. 2, WWW Server (second computer) intercepts / routes information to CGI stub for further routing to Dienst server (page server). Interception is triggered by a "Dienst request" (p. 11) packaged within a path portion of the HTTP request. When the HTTP request contains a "Dienst request" the CGI stub program is invoked.

[c] a dispatcher receiving said routed request from said interceptor and dispatching said request to said page server.

Dienst: (Id.), dispatching request to a page server is a feature of CGI-stub and thus it functions as a "dispatcher." See also discussion of "dispatching" claim 1[b6], supra. See Fig. 2, WWW Server (interceptor) routes request to CGI Stub. CGI Stub (dispatcher) dispatches request to Dienst Server (page server). (§ 4.1 and 4.2 at pp. 11-12.)

Claim 11:

[a] A machine readable medium having stored thereon data representing sequences of instructions, which when executed by a computer system, cause said computer system to perform the steps of:

Dienst: Pp. 6, 9-11, 30-33 (computer system comprises WWW server which includes a software Web server process such as CERN or NCSA Web server plus Dienst Server which is a multi-threaded software processes, such processes are inherently sequences of instructions stored on a machine- readable medium). See Fig. 2.

[bl] routing a dynamic Web page generation request from a Web server to a page server,

Dienst: Pp. 10-11; Fig. 2; Web page requests are routed from WWW server (Web server) to Dienst Server (page server), via a CGI Stub. The "Dienst request" packaged within the pat

portion of the HTTP request to the Web server (p. 11) triggers the routing of the dynamic Web page generation request (Dienst request) from a Web server to a page server (Dienst server).

[b2] said page server receiving said request and

Dienst: Fig. 2, pages 10 and 11, Dienst Server (page server) receives request.

[b3] releasing said Web server to process other requests wherein said routing step further includes the steps of

Dienst: P. 11, CGI stub strips Dienst request from HTTP request; the CGI stub is invoked each time the gateway Web server receives a Dienst request. Inherently, the WWW server is released from processing that request, and can process other requests. This is further apparent from the nature of the Dienst system to provide for simultaneous access to documents on the Dienst system for distributed WWW clients. The CGI stub accepts the request and forwards it to the Dienst server. Inherently, the Web server process is released from processing that request and can process other requests. "A Dienst server is a standalone multi-threaded process written in Perl." (§ 4.1 at p. 11.) A Web server that serves as a front-end for Dienst...", p. 11.

[b4] intercepting said request at said Web server,

Dienst: See Fig. 2, The Web server recognizes/ detects a path portion of an HTTP request, intercepts the request, and diverts it to a CGI stub.

[b5] routing said request from said Web server to a dispatcher, and

Dienst: Fig. 2, pp. 10-11, The HTTP request, detected by the Web Server is routed to the CGI Stub. The Dienst CGI stub (dispatcher) further sends request via I/O socket to Dienst Server.

[b6] dispatching said request to said page server;

Dienst: P. 11; CGI-stub dispatches request to a Dienst Server (page server). (§ 4.1 and 4.2 at pp. 11- 12.)

[c] processing said request, said processing being performed by said page server while said Web server concurrently processes said other requests; and

Dienst: Fig. 2, pp. 10-11; the function of the Dienst Server is to process requests and generate Web pages. Inherently, with the CGI stub program intercepting the Dienst request, the Web server process is released from processing that request, and can process other requests concurrently. The Dienst server processes the Dienst request. Because a "Dienst server is a standalone multi-threaded process" (§ 4.1 at p. 11), it necessarily processes the request while the Web server continues to concurrently process other HTTP requests. This is further apparent from

nature of the Dienst system to provide for simultaneous access to documents on Dienst system for distributed WWW clients. "A Web server that serves as a front-end for Dienst...", p. 11.

[d] dynamically generating a Web page, said Web page including data retrieved from one or more data sources.

Dienst: (Abstract; pp. 6, 9-12); Fig. 2; Dienst server dynamically generates Web page, retrieves data from data sources, such as documents on Dienst server or in a document database. See also Figs. 4 and 5, pp. 14-15 (showing dynamically generated pages); p. 58 (describing information that can be provided on dynamically-generated pages). See pages 57-61 regarding types of responses generated (generated Web pages) following a request. Because each search can be different, the process of responding to search requests necessarily is "dynamic."

Claim Rejections - 35 USC § 103

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

The factual inquiries set forth in *Graham* **v.** *John Deere Co.*, 383 U.S. 1, 148 USPQ 459 (1966), that are applied for establishing a background for determining obviousness under 35 U.S.C. 103(a) are summarized as follows:

1. Determining the scope and contents of the prior art.

2. Ascertaining the differences between the prior art and the claims at issue.
3. Resolving the level of ordinary skill in the pertinent art.
4. Considering objective evidence present in the application indicating obviousness or nonobviousness.

Claims 5 and 6 are rejected under 35 U.S.C. 103(a) as being unpatentable over Dienst, in view of Derler.

Claim 5:

The computer-implemented method in claim 3 wherein said step of processing said request includes the step of logging into said one or more data sources.

Dienst: Fig. 2, The Dienst Server "logs into" one or more data sources (other Dienst Servers or Document Database) § 8.3.1 at p. 41, "The basic Dienst server interoperates with a distributed network of other Dienst servers…A user may submit searches to a number of publishers, which are indexed by the distributed sites. A search is processed in parallel by each of these sites (logged into one or more data sources) and uniform search results (processed request) are returned to the user. The foreign server package enhances searching by allowing a Dienst server to interoperate with non-Dienst technical report servers (logged into one or more data sources). (inherent to accessing documents from foreign server is logging into the foreign server)."

Even more explicitly, Derler discloses "logging into data sources."

Derler: § 3.3.3 at pp. 35-36 (access rights dictated by username and password, inherently teaches "logging into" one or more data sources); § 4.5.6.1 at pp. 76-77, "…user has an account…logged

in as an identified Hyper-G user, which can be verified…Identification extends the user's access rights…"

Modification of the Dienst system to include securely logging into a data source as part of processing of requests, as taught by Derler, would be an obvious feature to access distributed documents.

Claim 6:

The computer-implemented method in claim 3 wherein said step of dynamically generating said Web page includes the step of maintaining a page cache containing said Web page.

Dienst discloses (p. 6 & 17) thumbnail images (page cache containing said Web page). The thumbnails are implemented using the HTML imagemap mechanism.
More explicitly Derler discloses caching of Web pages in a Web server to improve performance. (§ 3.3.5 at p. 37.)

Maintaining a page cache containing a Web page would be an obvious modification of the Dienst system. A person skilled in the art would have been motivated to add page caching in the Dienst server so as to provide ready access to frequently-requested documents without having to query foreign servers, as explained in the Derler Thesis at 3.3.5.

The following rejections are incorporated by reference as follows:

Claim Rejections - 35 USC § 102

(e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

Claims 1-11 are rejected under 35 U.S.C. 102(e) as being anticipated by USPN 5,701,451 to Rogers et al. See 90 / 008342, RXOSUB.R (11/27/2006) p. 3-7.

Claims 1-3, 5, and 7-11 are rejected under 35 U.S.C. 102(e) as being anticipated by USPN 6,249,291 B1 to Popp et al. See 90 / 008574, RXOSUB.R (04/03/2007) p. 73-78.

Claim Rejections - 35 USC § 102

(a) the invention was known or used by others in this country, or patented or described in a printed publication in this or a foreign country, before the invention thereof by the applicant for a patent.

Claims 1-5 and 7-11 are rejected under 35 U.S.C. 102(a) as being anticipated by Illustra. The rejections are incorporated by reference to 90/008574, RXOSUB.R (04/03/2007), p. 50-55)

Claims 1-5 and 7-11 are rejected under 35 U.S.C. 102(a) as being anticipated by Clausnitzer. The rejections are incorporated by reference to 90/008574, RXOSUB.R (04/03/2007), p. 61-66)

VI. Conclusion

In order to ensure full consideration of any amendments, affidavits or declarations, or other documents as evidence of patentability, such documents must be submitted in response to this Office action. Submissions after the next Office action, which is intended to be a final action, will be governed by the requirements of 37 CFR 1.116, after final rejection and 37 CFR 41.33 after appeal, which will be strictly enforced.

Any paper filed with the USPTO, i.e., any submission made, by either the Patent Owner or the Third Party Requester must be served on every other party in the reexamination proceeding, including any other third party requester that is part of the proceeding due to merger of the reexamination proceedings. As proof of service, the party submitting the paper to the Office must attach a Certificate of Service to the paper, which sets forth the name and address of the party served and the method of service. Papers filed without the required Certificate of Service may be denied consideration. See 37 CFR 1.550(f)

Please mail any communications to:

Attn: Mail Stop "Ex Parte Reexam"
Central Reexamination Unit
Commissioner for Patents
P. O. Box 1450
Alexandria VA 22313-1450

Please FAX any communications to:
(571) 273-9900
Central Reexamination Unit

Please hand-deliver any communications to:
Customer Service Window

Attn: Central Reexamination Unit
Randolph Building, Lobby Level
401 Dulany Street
Alexandria, VA 22314

Any inquiry concerning this communication or earlier communications from the examiner, or as to the status of this proceeding, should be directed to the Central Reexamination Unit at telephone number (571) 272-7705.

Mary Steelman
Primary Examiner

AU 3992

Conferees:

JOF
ESK

A11



Patricia Seybold Group

Trusted Advisors to Customer-Centric Executives

Three-Tier Client/Server Architecture

Achieving Scalability, Performance, and Efficiency in Client/Server Applications

By Wayne W. Eckerson
Patricia Seybold Group

UNAUTHORIZED REDISTRIBUTION OF THIS REPORT IS A VIOLATION OF COPYRIGHT LAW

P.O. Box 290565, Boston, MA 02129 • Phone 617.742.5200 • Fax 617.742.1028 • www.psgroup.com

FEATURED REPORT:

BY WAYNE W. ECKERSON

Three-Tier Client/Server Architecture

Achieving Scalability, Performance, and Efficiency in Client/Server Applications

Introduction: Setting the Stage

Architecture Is Critical

The most hardened mainframe bigot has come to realize that client/server computing is not just another passing fad in the computer industry. Client/server computing is here to stay. The challenge now facing users is not whether to adopt client/server computing but how to make it work effectively to solve business problems on an enterprise scale.

Most client/server applications today are departmental in scope. They are built on a two-tier architecture that is designed to support 15 to 20 users and run non-mission-critical functions. Most of these applications have been built with popular graphical user interface (GUI) development tools that pack all the code for the user interface, application logic, and services in a Windows PC. The client issues SQL calls across a local area network to a relational database to retrieve data for processing.

Encouraged by the successes of these initial client/server applications, companies have been quick to expand the scope and complexity of the applications to address pressing business opportunities or requirements. Managers have asked developers to increase the number of users, functions, and data sources supported by these applications. This inexorable "scope creep" has been the downfall of many early client/server applications.

Developers are now recognizing that a two-tier architecture is not sufficient to support enterprise client/server applications that provide high performance, scalability, availability, and reliability.

Middle Tier of Application Servers

THREE-TIER MODEL. Enter the three-tier client/server architecture. The three-tier architecture extends the previous model by adding a middle tier of intermediate servers to support application logic and distributed computing services. The middle tier is critical for providing location and migration transparency of distributed

resources as well as a variety of services required to provide reliable, secure, and efficient distributed computing.

Companies planning to build more scalable client/server applications should consider deploying a three-tier architecture. A three-tier architecture will allow companies to leverage their investments in client/server to create applications that provide real bottom-line payback.

However, building three-tier applications isn't easy, and there is little real-world experience to shed light on the process. Several of the programming tools that shipped in the past year support the design and deployment of three-tier applications, but these tools are still immature and don't provide all the services needed to support a distributed computing environment.

This article analyzes the makeup of a three-tier client/server architecture and examine the benefits and challenges it offers. It also examines the tools available today for building three-tier client/server applications and the key features and services they should support. Finally, the report examines how three-tier client/server architecture lays the foundation for distributed object computing.

Definition of Terms

Application Components vs. Physical Platforms

As with any new buzzword in the computer industry, three-tier architecture means slightly different things to different people. The precise definition of tiers and what they represent in user computing environments are fuzzy.

A three-tier architecture has two aspects. One is the *application architecture*, and the other is the *deployment architecture*. Application architecture refers to the constituent parts that make up an application. These are presentation, functional logic and data. Together, these components comprise the logical representation of an application.

A deployment architecture, on the other hand, specifies the physical configuration of computers on which an application runs. In a two-tier deployment architecture, the client portion of the application runs on a desktop PC or workstation, and the server portion runs on a server machine across a network. The workstations and server machines are considered the *tiers* in a deployment architecture.

A three-tier deployment architecture inserts an intermediate server between the desktop PC and the central server. In a three-tier architecture, the desktop PC handles the application's presentation component, the intermediate server supports functional logic and services, and the back-end server handles data processing. Here the physical deployment environment mirrors the logical application architecture. This creates an inherent synergy that provides many benefits. (See Illustration 1.)

© 1995-2007 Patricia Seybold Group
Reproduction in whole or in part of this report is prohibited. • 2
All unauthorized redistribution is a violation of copyright law.

Definition of Terms

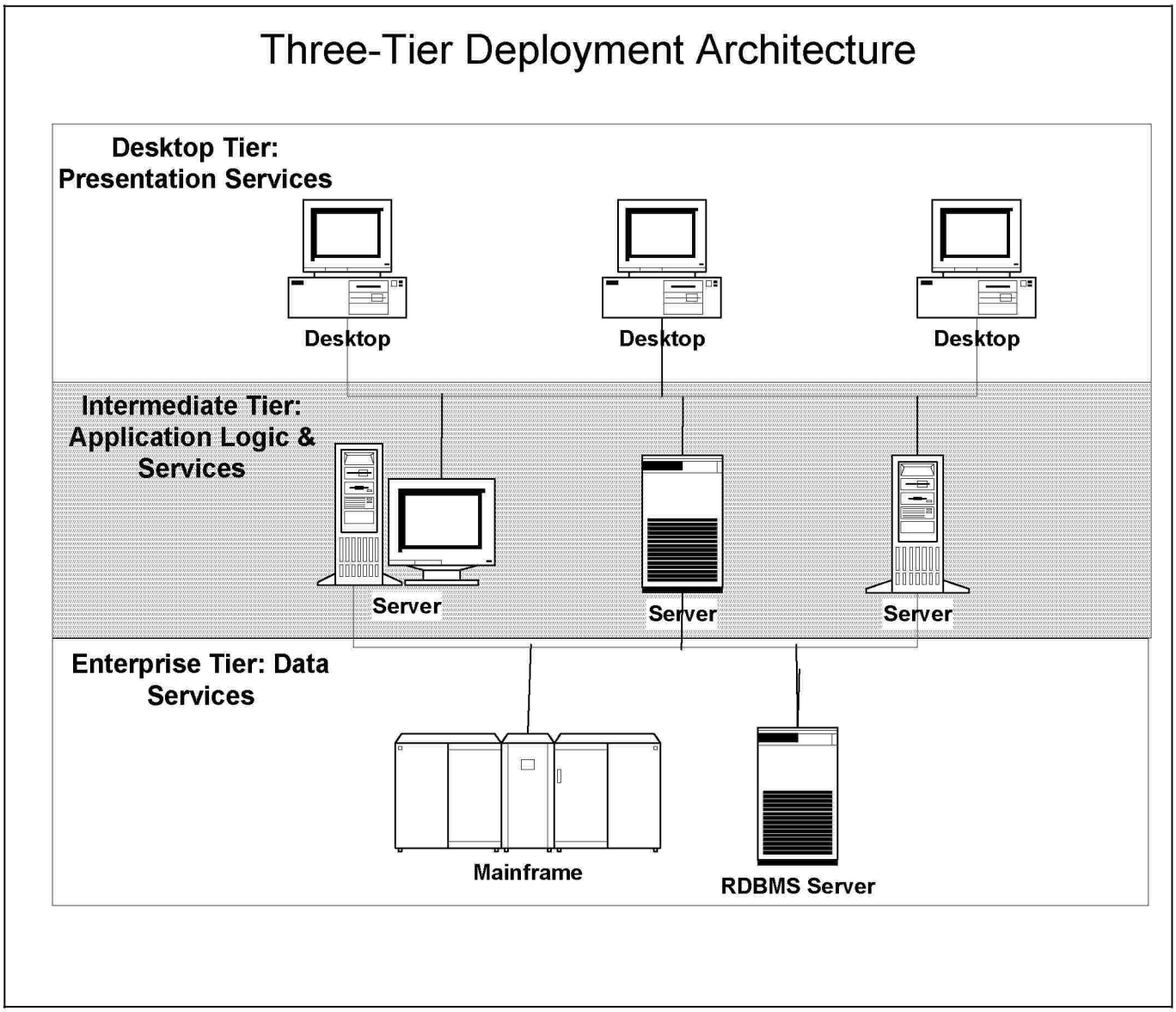


Illustration 1. A three-tier architecture deploys application components (presentation, functional logic, and data) across three tiers of computer platforms: desktop machines, intermediate application servers, and back-end database servers.

Customers.com® Research Service
© 1995-2007 Patricia Seybold Group

Reproduction in whole or in part of this report is prohibited. • 3
All unauthorized redistribution is a violation of copyright law.

Application Architecture

Application Architecture

Application Components

As was mentioned above, the three basic elements of an application are presentation, functional logic, and data. *Presentation* refers to the user interface, or the part of an application that controls what is displayed on a user's workstation screen. *Functional logic* refers to the tasks and rules that govern the way a business operates and the services required to implement them. Business rules are embedded in programming code. *Data* refers to the information the business accumulates about its customers and operations as well as the services required to access and manage those data.

DECOMPOSABILITY. The key to three-tier application architecture is to ensure that each of these application elements is a separate, independent component within the application. Another way to say this is that three-tier applications can be decomposed into presentation, functional logic, and data. This means that each component can be modified or replaced without any of the other components being rewritten or changed.

Many Applications Aren't Decomposable

Most older mainframe applications and some early client/server applications are not decomposable. Instead of being logically separated, these components are inextricably intertwined in the fabric of the application. Consequently, application components can't be easily extracted and ported to another platform or changed without affecting other parts of the application. It is difficult, for example, to migrate these types of mainframe applications to client/server platforms. Either the code is ported in entirety to Unix, or the applications have to be rewritten from scratch.

GUI Builders. Client/server applications developed with popular GUI builders, such as Powersoft's PowerBuilder, link presentation and logic and rely on a remote server to store and manage data. These tools let developers paint application screens with GUI objects, such as radio buttons and dialog boxes, and then attach scripts to each object. Clicking on the object activates the script, creating an event-driven program. The logic in these applications is tied to the GUI. Changing the logic requires rewriting a script and relinking the script to the appropriate GUI object.

In a decomposable application, presentation and function are separate, independent entities. This allows developers to run presentation on a desktop PC and functional logic on one or more computers on the network, which makes it easier to update applications since the logic is located in one place instead of on hundreds of remote computers.

Application Interfaces

There are three ways to create decomposable applications. One is to use a traditional programming language to segment monolithic applications into callable procedures. Another is to link programs running presentation, logic, and data components via common interfaces, such as remote procedure calls (RPCs) or application programming interfaces (APIs). The third way is to build presentation, function, and data components from a series of autonomous objects. (See Illustration 2).

Customers.com® Research Service
© 1995-2007 Patricia Seybold Group

Reproduction in whole or in part of this report is prohibited. • 4
All unauthorized redistribution is a violation of copyright law.

ORCL01043754

Application Architecture

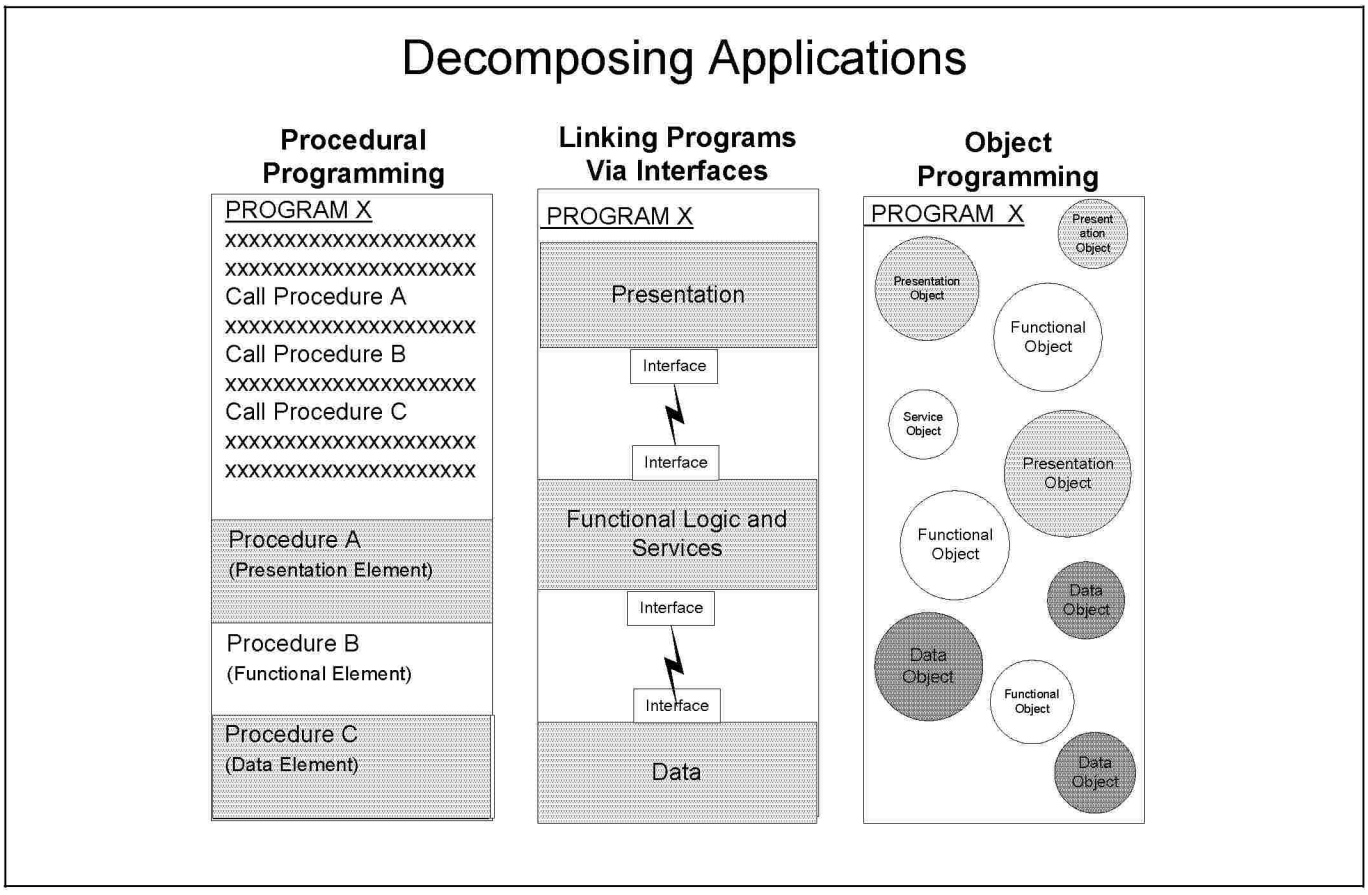


Illustration 2. The diagram depicts three methods for creating decomposable applications. Decomposing applications is the first step in creating three-tier applications.

Application interfaces can range from SQL APIs, such as Microsoft's Open Database Connectivity (ODBC) and Sybase's DB-Lib, to RPCs to vendor-proprietary APIs.

Building Interfaces between All Application Components May Be Overkill

For example, middleware vendor Open Environment has helped evangelize the notion of three-tier architecture. Its tools enable developers to build interfaces between presentation, function, and data service components of an application using a proprietary RPC or the Open Software Foundation's (OSF's) Distributed Computing Environment (DCE) RPC. However, building RPC interfaces between all application components may be overkill for many client/server applications. The method begins to pay off as companies begin to create enterprise environments in which large numbers of clients need access to lots of distributed services in order to execute applications.

Hybrid Interfaces. A more common approach is to leverage vendor interfaces to link up components that may be distributed across platforms. For example, 4GL vendor Magna provides a transactional API on Windows PCs that enables developers to integrate front-end GUI programs, such as those written with Powersoft's

Reproduction in whole or in part of this report is prohibited. • 5
All unauthorized redistribution is a violation of copyright law.

Application Architecture

PowerBuilder, to the 4GL Magna code that resides on a server. Magna also provides an API to execute transactions on relational database management systems (RDBMSs). Developers can also use the API provided by their RDBMS vendor to access data stored that database.

Component-Based Software

Procedural Programs. Aside from interfaces, developers can create decomposable applications within a single programming language. This requires up-front planning and discipline. With traditional programming languages, developers must define portions of their code to be segmented into callable procedures. These procedures can be ported to another platform and connected back via an RPC at a later date. Most new mainframe applications are being designed in this way, which makes it easier to migrate these applications to client/server portions. Developers can simply call out the presentation and logic procedures and port them to a client/server platform. Unfortunately, these applications can gradually lose their decomposability as patches are applied. Companies must be diligent about maintaining a strict separation of application components.

Objects Extend Decomposability

Object-oriented applications extend the decomposability of applications to much finer levels of granularity. Objects can represent presentation, functional logic or data, or smaller elements within each of these components. For example, an object might be a radio button on a GUI or an "account summary" command in a bank teller application. Because objects encapsulate their implementation behind a generic interface, moving them from one platform to another is easier. Swapping out the underlying program that executes the object's behavior or methods is also easier.

Three-Tier Deployment Architecture

Enterprise Applications Often Require Three Tiers

Companies have discovered the hard way that client/server applications built on two-tier deployment architectures don't scale well enough to support enterprise applications. By "enterprise," we mean applications that support core operational systems that run the company, such as order entry, inventory, distribution, and finance.

Enterprise Application Characteristics

These bet-your-business applications typically have been run on mainframes and minicomputers, and they comprise millions of lines of code and hundreds of gigabytes of data. The applications are complex, involving considerable up-front analysis and design. Traditionally, completing a program has required 50 to 100 developers working for a year or more. The applications are used by hundreds or even thousands of users spread across large geographic areas. Enterprise applications have requirements for high reliability, availability, portability, scalability and performance.

Early Stages of Deployment

These enterprise applications requirements are difficult for client/server applications to meet. Many leading-edge users claim that only a three-tier client/server architecture can achieve these requirements. However, we are still at an early juncture in the deployment of three-tier client/server applications. There is little

© 1995-2007 Patricia Seybold Group

All unauthorized redistribution is a violation of copyright law.

empirical evidence to determine whether three-tier architectures or client/server computing in general are up to the task. The few companies that have deployed three-tier client/server applications have done so in support of decision-support tasks. Most companies are still running operational applications in terminal-host environments, whether the dominant platform is a mainframe, minicomputer, or Unix server.

Three-Tier Architectures Are Useful for Downsizing

Ironically, many companies view the three-tier architecture as the most convenient way to migrate from terminal-host to client/server computing environments. These companies are moving presentation to intelligent workstations and either porting or rewriting mainframe applications to run on intermediate servers. Many of them plan to eventually migrate data off the mainframe once there are sufficiently high-powered Unix systems and RDBMSs. For now, many are content to use mainframes as giant data servers. Some companies, however, are moving all application components to smaller computing platforms.

The Many Roles that Intermediate Servers Play

Add Flexibility in Application Configuration

A three-tier deployment architecture leverages the components of a decomposed application. Since application elements are independent modules linked via some sort of interface, this gives developers or architects the freedom to deploy these components on optimal platforms.

For example, NASA may have an application that helps plot voyages of the space shuttle. Part of that application may involve computing the trajectory of the spacecraft at liftoff to obtain optimal orbit during a week-long voyage. This compute function might perform better if it were calculated on a high-performance workstation or supercomputer. A three-tier application would allow that function to run on a separate intermediate server. Moreover, it would make the function available to current and future applications that may need to perform similar calculations.

A three-tier architecture allows users to create a middle tier of intermediate servers whose purpose is to process services or functions shared by many applications. Intermediate servers inject scalability and reusability into client/server environments. Instead of writing the same function or service in every new application, developers can write the application once and place it on a server accessible by all applications.

Data Warehouses

Similarly, companies can use intermediate servers for staging mainframe data. Companies could regularly extract mainframe data pertinent to a given department and download the data to an intermediate server. This moves data closer to users who need it, offloads processing cycles from the mainframe, and eases network loads.

Services Platforms: Gateways and Transaction Monitors

Intermediate servers can also support communications gateways, database gateways, and transaction monitors. For example, companies can run a database gateway, such as Information Builders' EDA/SQL, on an intermediate server to give users transparent access to more than 50 SQL and non-SQL data stores across a network. EDA/SQL translates client API calls to the appropriate syntax for accessing remote

© 1995-2007 Patricia Seybold Group
Reproduction in whole or in part of this report is prohibited. • 7
All unauthorized redistribution is a violation of copyright law.

data sources and handles underlying communications. Application developers don't have to know the details of where remote data sources are implemented and how to access and navigate through them to find specific data.

MIGRATION TRANSPARENCY HIDES PLATFORM CHANGES FROM USERS. Besides location transparency, gateways running on intermediate servers provide migration transparency. That is, users can change the location or implementation of back-end resources without having to rewrite front-end applications. Intermediate servers allow companies to detach client applications from back-end resources.

Consider, for example, a company that wants to migrate an RDBMS that is reaching its capacity from a uniprocessor to a symmetric multiprocessing machine on another part of the network. The address change is made once in the database gateway, and all client requests are redirected to the right destination. Clients still use the same interface calls to request data services.

Transaction Monitors Provide Critical Application Services

FAILOVER, LOAD-BALANCING. Companies are increasingly deploying transaction monitors on intermediate servers to augment their RDBMSs with key services, such as failover, load-balancing, and rollback and recovery of transactions. Transaction monitors, such as AT&T Global Information Solutions Top End or Novell's Tuxedo, manage the flow of requests to application resources, which are typically, but not exclusively, relational database engines. The transaction monitor can balance fluctuating loads across multiple instances (servers) of an application resource. It can also sense when an application server has failed and automatically redirect requests to remaining available servers. This provides high availability of system resources.

PRIORITIZATION. Transaction monitors can also enforce a prioritization scheme, in which high-priority requests are given immediate access to application resources while low-priority requests are throttled back. This feature can help companies prevent users issuing ad hoc SQL queries from bogging down an online transaction system that drives the company's core business.

TRANSACTIONS. Finally, transaction monitors provide two-phase commit capabilities to ensure that updates across one or more databases are handled correctly and that data integrity is not jeopardized. Although many DBMSs provide commit, recovery, and rollback services, as well others mentioned above, transaction monitors running on an intermediate gateway provide a DBMS-independent method for implementing these services. Many companies are now using transaction monitors to manage all aspects of client/server interaction, not just for DBMS access and transaction control.

© 1995-2007 Patricia Seybold Group
Reproduction in whole or in part of this report is prohibited. • 8
All unauthorized redistribution is a violation of copyright law.

A12

Computer Networks and ISDN Systems 27 (1995) 1017–1026

COMPUTER NETWORKS and ISDN SYSTEMS

A WWW interface to the OMNIS/Myriad literature retrieval engine

Alexander Clausnitzer [a,1], Pavel Vogel [a,2], Stephan Wiesener [b,3]

[a] *Fakultät für Informatik, Technische Universität München, Orleansstrasse 34,D-8167 München, Germany*
[b] *Bayerisches Forschungszentrum für Wissensbasierte Systeme (FORWISS), München, Germany*

Abstract

Cataloguing and searching procedures in traditional library systems are expensive, time-consuming and often incomplete. OMNIS is a novel multimedia information retrieval system for the administration of documents in libraries and offices. Using the fulltext database system Myriad combined with scanning and OCR technologies it offers the disclosing, archiving and searching functions at drastically reduced costs with much more precision. Documents may contain page images, full-length PostScript or other medial information and offer the user a much better insight into documents. At Technische Universität München a considerable number of computer science documents have been made searchable by a simple fulltext query language. To make this document retrieval system available to WWW clients the OMNIS document access function was implemented as OMNIS-WWW server which is already in operation. This paper contains the substantial features of OMNIS (especially its searching function) and discusses the concepts and the implementation of its WWW server.

The development of OMNIS is promoted by DFG (Deutsche Forschungsgemeinschaft) and DFN-Verein (Deutsches Forschungsnetz) [6].

Keywords: Digital libraries; Multimedia database systems; Information retrieval; Fulltext search; Document archiving

1. Introduction

The human knowledge still is mostly conserved and distributed by printing it on paper. Libraries and other institutions buy books, journals, etc. and try, with moderate success, to disclose their contents by manual indexing and classification. With about $40.00 per book the indexing and cataloguing costs are high and consume the main part of the library budget. This is why most catalogs can offer incomplete bibliographic information only. An efficient use of these knowledge sources is connected with several problems: using manual catalogs literature search is time-consuming and often inaccurate. Beyond this, the selected documents are not immediately available and have to be ordered and physically transported to the reader. The library system OMNIS/Myriad [4,7] is a tool for an efficient document management. Its architecture consists of two layers: OMNIS as a library application and Myriad [20] as the underlying

[1] clausnia@informatik.tu-muenchen.de; http://sunbayer35.informatik.tu-muenchen.de/~clausnia
[2] vogel@informatik.tu-muenchen.de; http://www3.informatik.tu-muenchen.de/public/mitarbeiter/vogel.html
[3] wiesener@forwiss.tu-muenchen.de; http://www.forwiss.tu-muenchen.de/~wiesener/public/user.html

0169-7552/95/$09.50 1995 Elsevier Science B.V.
SSDI 0169-7552(95)00026-7

ORCL01806610

Table 1
OMNIS databases offered at TUM

Library	Contents	# Docs
ZentralBib	Central library catalog	195 000
InfoMathBib	Department of Computer Science library catalog, from August 93 on with abstracts, tables of content, and scanned pixel images	40 800
TUBibMue	Books, journals, conference proceedings, and technical reports on computer science (1981–1992), bibliographic data with abstracts, no pixel images	125 000
TechRep	Generated from BibTex files of computer science literature from over 100 universities	209 000
BayerBib	Literature on database management systems, knowledge-based systems, and information systems in general, all with abstracts, tables of contents and pixel images	3285
MayrAlphaBib	Literature on theoretical computer science, partly with abstracts, tables of contents, and pixel images	30 000
WiederholdBib	Literature on database management systems, bibliographical data partly with abstracts, tables of contents, and images	20 000

fulltext database system. OMNIS has been developed at the Computer Science Department of Technische Universität München and FORWISS München (Bayerisches Forschungszentrum für Wissensbasierte Systeme).

The two basic ideas of OMNIS are:

(1) Replacing the manual indexing and classification procedure by storing those parts of a document as fulltext which contain characterizing terms and phrases (abstract, table of contents, etc.). The document retrieval can then be done by fulltext queries.
(2) Storing at least the characterizing pages as pixel images or PostScript if available to offer it to the user in a formatted way which may contain additional graphical information and is much better legible than a plain ASCII text.

For archiving a document (or parts of it) it is scanned in as a pixel image. Then characterizing text and optional bibliographic data (title, authors, publisher, etc.) are converted by commercial OCR software into text format. The bibliographic information can easily be selected from the fulltext part via cut-and-paste and is stored into the OMNIS database together with picture and fulltext data. This way the archiving costs can be reduced to $1.50 per document.

After one year of operation at Technische Universität München (TUM) a considerable amount of documents has been made searchable in OMNIS digital libraries. Table 1 lists some of them (state November 94), which makes a total of more than 622 000 documents. At TUM OMNIS became an integral part of any research activity. In November 1994, 337 different users made 2156 literature search sessions. As we know, OMNIS databases in Munich are the most comprehensive collection of computer science documents and new publications are added daily. Compared to other literature retrieval systems, OMNIS documents may additionally contain BLOBs (Binary Large Objects) [17]. A BLOB may hold images, i.e., a few characteristic pages in some image format as described above, whole PostScript [2] documents or audio/video data. To make the OMNIS literature search available to the whole science community a **WWW-based OMNIS server** [4] was developed. WWW has been chosen because it is well known to the science community. It can mediate access to OMNIS without expensive distribution of special retrieval clients. WWW supports documents with images, audio and video data and offers the best user interfaces of common wide area information retrieval systems.

In Section 2 an overview of the whole OMNIS library system and especially of its search facility is given. Afterwards the retrieval component is dis-

[4] http://www3.informatik.tu-muenchen.de/cgi-bin/omnis

ORCL01806611

cussed in more detail. Section 3 describes the concepts of realization and the implementation of a WWW-based retrieval component. Section 4 is a short summary.

2. Structure of the OMNIS system

OMNIS has primarily been developed as an administration and retrieval tool for scientific libraries and other collections of scientific documents in paper form.

It consists of archiving, searching, and lending components. This partition is predefined by the three classes of systems users: people archiving documents, users performing literature search, and professional library personnel, responsible for library operation. The system's workflow is given in Fig. 1.

2.1. OMNIS / Myriad architecture

OMNIS/Myriad is designed as a distributed system in a client/server architecture. Any number of OMNIS servers can be installed and each server may manage multiple document databases. A special OMNIS client offers retrieval and/or the archiving function to the user. A rough scheme of this architecture is shown in Fig. 2.

To realize a WWW-based literature search in OMNIS databases a special searching component has to communicate with the user's WWW clients and treat the OMNIS database servers as usual. Viewed from the side of the WWW system such a component can be called the *OMNIS-WWW* server. Viewed from the rest of the OMNIS system this component has to be located at the same level as the normal OMNIS searching component, i.e., as a client.

2.2. OMNIS documents

An OMNIS document in the database always consists of some structure fields (i.e., bibliographic data like title, author, publisher, etc.) and a piece of text containing the characteristic expressions and phrases of the document (e.g., abstract, table of contents, etc.). The structure fields are subject to structure (relational) queries while the text is subject to (associative) fulltext queries. The contents of the

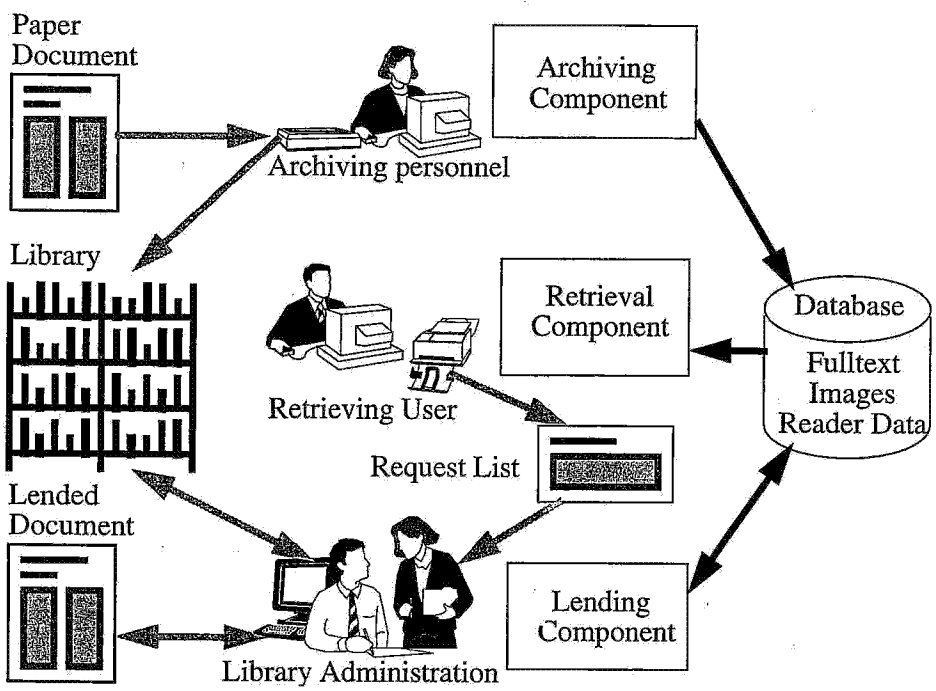


Fig. 1. OMNIS system components and user groups.

ORCL01806612

structure fields are attached to the text automatically and so the words and phrases contained in the structure fields can be found by both structure and fulltext queries.

Additionally, a number of BLOBs containing medial information can be attached to an OMNIS document. Frequently used are image BLOBs containing scanned-in pages. These pixel images are the basis for an OCR procedure which produces the document's fulltext mentioned above. Further PostScript BLOBs containing complete documents may be attached. The presence of at least a few pages in original format proved to be essential for the users, not only because of possibly contained graphical information but also because of the clear, convenient and familiar outward appearance. Audio and video BLOBs are possible too. For a library application, however, only a few original documents with audio and/or video data are available yet. The medial information is stored as part of a document and can be presented to the user. BLOBs are not interpreted and thus, not subject to querying [5].

2.3. Document retrieval

For efficient literature search and high acceptance by end users a simple and intuitive query interface is necessary. Such a tool is the OMNIS fulltext query language. In the following some examples give a rough idea of this language:

`1. gaus`	single word query
`2. database machine`	phrase query
`3. h_permedia%`	character wildcards
`4. live cycle & (basili | cohen)`	boolean operators
`5. #2 & boral`	reference to a former query
`6. fulltext .. database`	word wildcards
`7. ?author= 'Boral% '`	structure field query

The first query delivers all documents containing the word "gauss" (or "Gauss") somewhere in their fulltext. The second one similarly searches for the phrase "database machine". In the third, sixth and seventh query wildcards stand for any single characters, character sequences or single words. Fulltext queries are simple to formulate and proved to have a good precision. Query four shows the usage of boolean operators to formulate more complex needs whereas successive query refinement is possible by referencing former requests as shown in query five. Query seven results in documents whose structure field "author" begins with "Boral".

3. The OMNIS-WWW interface

In the last two years WWW (World-Wide Web) has become one of the fastest spreading Internet applications. This success may have two main reasons:

(1) The prevailing hypertext systems seem to be restricted to one local or distributed file system. Existing wide area information retrieval and ac-

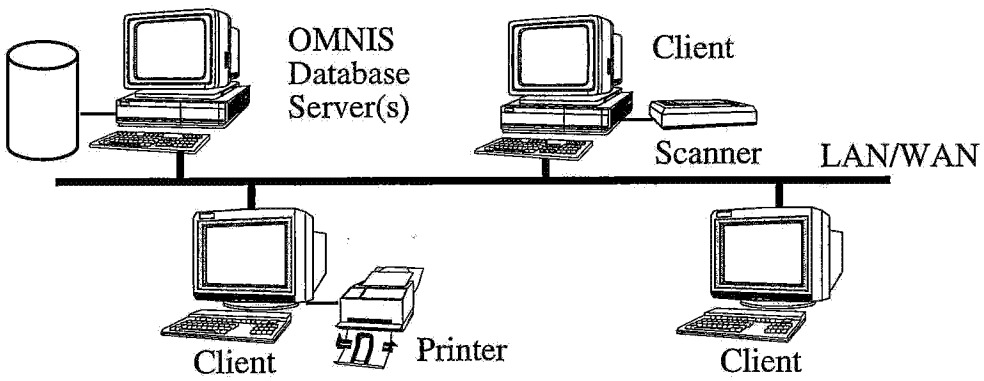


Fig. 2. OMNIS client/server architecture.

ORCL01806613

cess systems like FTP [19], Gopher [3], Telnet, WAIS [1], News [16], and x500 miss the hypertext easy navigation facility. The basic WWW idea, the merge of hypertext and wide area networking, has closed a gap perceived by many users. Additionally, using an addressing scheme which is compatible to the wide area systems above (with additional link facility), their documents are available too.

(2) WWW clients are freely available for most existing platforms and terminal types. On terminals with graphical user interface it provides a complete point and click interface of high user acceptance. The retrieved information pages may also contain images and other medial data. A WWW page can also be displayed on a plain text terminal with the only restriction that images are replaced with an alternative text.

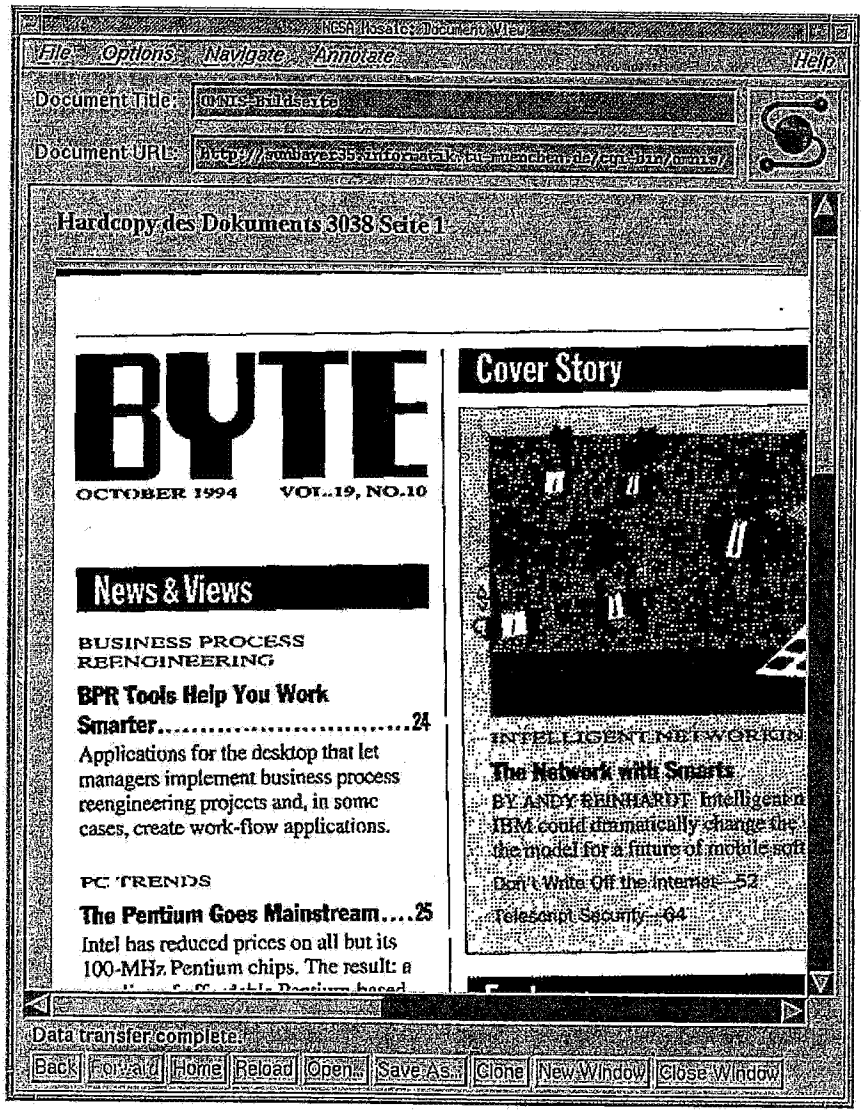


Fig. 3. An open OMNIS document displayed by XMosaic.

ORCL01806614

3.1. Substantial WWW features

WWW offers the retrieval of hypertext pages written in HTML (Hypertext Markup Language) [8,13] by the Hypertext Transfer Protocol (HTTP) [9]. A HTML hypertext page may contain links to any document in the WWW addressing space [10]. In addition to a simple document retrieval, HTTP includes several not yet generally used features for automatic accounting, user identification, and encryption. (See Fig. 3.)

The WWW client sends a request to an HTTP server which sends the requested data back to the client. Besides describing ordinary hypertext pages HTML is able to define forms, which can be filled out by the user. To use the entered data the HTTP server has to be able to process it.

Instead of mapping an URL onto a file in the local file system of the HTTP daemon, servers like *cern-http* from CERN and *httpd* from NCSA [18] have the capability to execute applications if certain URL addresses are requested. These gateway applications are ordinary executable programs which can be written in any programming language (e.g., C , Perl, or shell scripts). The communication between HTTP server and gateway application is defined in the CGI (Common Gateway Interface) definition [18]. Prior to the start of a gateway program, the HTTP server writes data about the client's request (e.g. origin, variables sent by the client, and requested address) into environment variables or StdOut of the operating system. These data can be read and processed by the gateway application to produce output in MIME format [11] which is sent back to the HTTP server's StdIn.

Improvements of HTTP and HTML for requests at library databases or for coding information like title, author, publisher, etc. are currently under discussion. A final standard is not to be expected in the near future.

3.2. Solution alternatives

In contrast to client/server relationships in usual database systems HTTP only offers a stateless connection. No permanent connection between client and server during a complete session is established. This raises several problems for an OMNIS gateway to WWW.

For every incoming HTML request with database access a new database session has to be opened and closed again after the data transfer is completed. Because of OMNIS internal reasons the login and logout procedures are time-consuming functions. Opening and closing sessions for each single request would make efficient working with the WWW interface impossible. Nevertheless, HTTP will provide mechanisms to control virtual sessions in the future. These features, however, are not yet supported by most WWW clients.

A further problem of closing database sessions after every client's request is the difficulty of re-using information which already had been accessed in a former request. This requires that the actual session status must be stored in HTML forms of the type `hidden` or within URL addresses.

To solve these problems two solutions were under discussion:

(1) Development of a new OMNIS-HTTP server being based on a WWW library of common code [14] developed by CERN. Because of OMNIS internal reasons for each OMNIS database an extra OMNIS-HTTP server with a different HTTP address has to be started. Each OMNIS-HTTP server is in permanent connection with its OMNIS database and translates incoming requests into database queries. The resulting data is then transformed back into HTML and returned to the WWW client (see Fig. 4).

(2) Using an existing HTTP server which calls gateway applications (see Section 3.1). Such an OMNIS-HTTP gateway application consists of a stand alone OMNIS server for each database and a single CGI program. Each OMNIS server has a permanent connection to a single database within a permanent database session. The CGI program passes the WWW client's requests to one of the

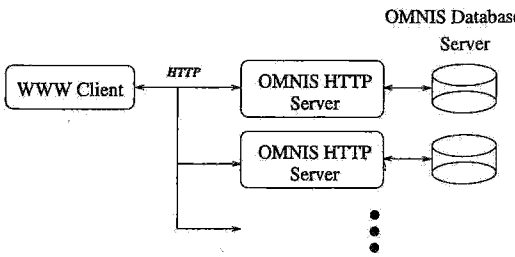


Fig. 4. Independent OMNIS-HTTP servers.

ORCL01806615

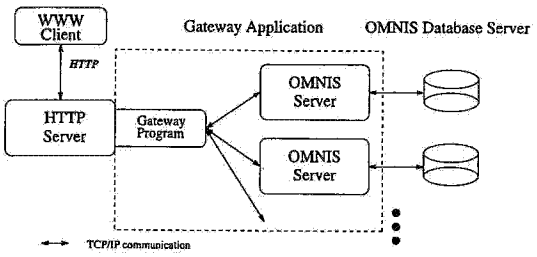


Fig. 5. OMNIS-HTTP gateway application.

OMNIS servers which translate them into OMNIS queries for the database. The query results are sent back to the CGI program which transforms it into a WWW client readable format (see Fig. 5).

The advantage of the first solution is that all attributes of the HTTP protocol like user-authorization or multi-language support can be utilized. Additionally there is no need for a further OMNIS server. The disadvantage is the already mentioned need for different OMNIS-HTTP servers with own addresses for each database.

The second solution offers the capability to use one gateway program with the same base URL address for all databases. In the future an automatic load balancing of OMNIS servers for one database would be easy to implement. Existing HTTP servers already have built-in features for auditing and user authorization and it is not necessary for the application to deal with HTTP protocol details. Unfortunately the current CGI standard offers only limited information about the client's request to the gateway application. Within the next years the HTTP protocol will be improved. Since the OMNIS gateway program is a standard CGI application it will still be able to work with improved HTTP servers which can use new features like data encryption and automatic accounting.

Finally, the second solution was implemented [12].

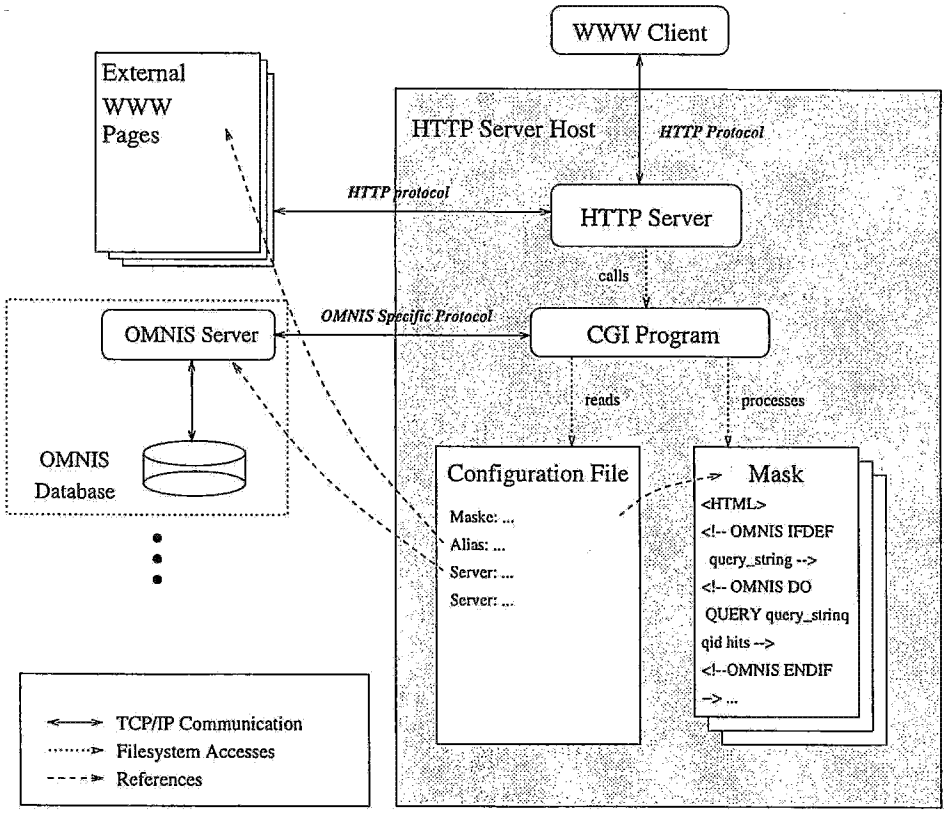


Fig. 6. Interplay of HTTP server, CGI program, and OMNIS server.

ORCL01806616

3.3. Solution concepts and implementation

A major point in the realization of the OMNIS-HTTP gateway was to make it easy to customize the pages and forms for different types of OMNIS databases. Since many databases should be available via WWW the gateway application has to provide easy methods for the administrator to add and change database parameters. For these purposes a special Mask Definition Language was developed. The masks consist of the fixed HTML fragments which are filled by the gateway application and returned to the client as request results. These fragments also contain hidden queries for the OMNIS server and control commands for the CGI program.

Retrieval requests from WWW clients are handled as follows (see Fig. 6):

(1) The HTTP daemon receives a HTTP request (e.g., a fulltext query) from a client. If the URL refers to a CGI program, it is executed (e.g., for query processing). The communication takes place as described in Section 3.1.

(2) The CGI program analyzes the client's request and reads a global configuration file. This file contains OMNIS server addresses and defines how resulting HTML pages are to be generated.

(3) This result is produced in one of three different ways, depending on the client's type of request:
   - The requested page is not really generated by the gateway program. A line in the configuration file is a reference to any existing page in the WWW addressing space. The gateway program causes the HTTP server to send a command to the client to load this specific page.
   - The requested page is generated by the gateway program. A line in the configuration file points to a mask which contains commands for the CGI program or queries for OMNIS servers. These masks have to be accessible in the filesystem and are parsed by the CGI program. Encoded queries and commands are executed. The resulting output is a HTML page which is transferred to the HTTP daemon and sent back to the client.
   - The requested page delivers binary data like an image, postscript, or plain text. These data are loaded from the OMNIS server and are transformed into common formats by the CGI program. The result in MIME format is given to the HTTP daemon and sent back to the client.

(4) The client presents the received data.

Demands on the mask definition language:

The concept of the mask definition language should allow reading and editing mask pages with ordinary HTML browsers or editors.

Not to be limited to the maximum command length and syntax of SGML (Structured Grammar Markup Language) [15] which HTML is based on, it was necessary to hide commands for the OMNIS gateway application in SGML comments. For this reason these commands start with `<!-- OMNIS` and end with `-->`. To extract information from the client's request (e.g., query string, query ID, database name, etc.), all data delivered by HTTP GET or POST requests and parts of the URL address are stored in internal variables of the OMNIS gateway application. These are used or modified by commands and can be inserted into any place of the resulting HTML document.

There are commands for the OMNIS server to start a query, get structure field information for one specified document or documents found by one specific query, get document-text, get a list of existing pictures and get other binary data.

In contrast to the CGI program the OMNIS server is only started once for each OMNIS library and handles incoming requests sequentially. The gateway program is executed for each client request. With fast following HTTP requests the CGI program can be started several times by the HTTP server to run parallel. To ease the OMNIS server, time-consuming functions like transformation of image formats from an OMNIS internal format to GIF is done by the CGI program.

4. Summary

In this paper the problem of high indexing costs in traditional library systems was disclosed. A short overview of the OMNIS digital library system's architecture was given and it was explained how

ORCL01806617

OMNIS document archiving and retrieval means solve the above problem.

During the last years, many OMNIS databases have been filled with huge amounts of scientific literature. To make these information sources accessible for the growing community of World-Wide Web users an OMNIS-WWW gateway was developed and is in operation today. The implementation is based on the CGI (Common Gateway Interface) definition for HTTP servers. A CGI program is connected to a HTTP server as well as to several OMNIS database servers. It distributes incoming client requests to OMNIS servers and returns their query results back to the HTTP server.

References

[1] B. Lincoln, WAIS Bibliography, Technical Report, Thinking Machines (August 1991). **ftp://quake.think.com/pub/wais/wais-discussion/bibliography.txt** [5]

[2] Adobe Systems Inc.: PostScript Language: Reference Manual. Adobe Systems Inc., 1991.

[3] F. Anklesaria et al., The Internet Gopher Protocol, Internet RFC 1436 (March 1993).

[4] R. Bayer, OMNIS/Myriad: Electronic Administration und Publication of Multimedia Dokuments. In: *Informatik, Wirtschaft und Gesellschaft 23. GI-Jahrestagung* (Springer Verlag, Dresden, 1993).

[5] R. Bayer, W. Kowarschick, Ch. Roth, P. Vogel, S. Wiesener: OMNIS/Myriad on its Way to a Full Hypermedia System, *EITC 94 (European Information Technology Conference), 1st Workshop on Human Comfort and Security*, June 1994, Brussels.

[6] R. Bayer, P. Vogel and H. Göttsch, The Munich Metropolitan Area high speed network of digital libraries, Internal Report, Technische Universität München, Munich (1994).

[7] R. Bayer, P. Vogel and S. Wiesener, OMNIS/Myriad Document Retrieval and Its Database Requirements, in: *DEXA 94 (Database and Expert Systems Applications), 5th International Conference, Proceedings* (Springer Verlag, 1994).

[8] T. Berners-Lee, Hypertext Markup Language (HTML) (January 1993). **http://info.cern.ch/hypertext/WWW/MarkUp/MarkUp.html** [6]

[9] T. Berners-Lee, Protocol for the Retrieval and Manipulation of Textual and Hypermedia Information (1993). **http://info.cern.ch/hypertext/WWW/Protocols/HTTP/HTTP2.html** [7]

[10] T. Berners-Lee, Uniform Resource Locators (January 1993). **http://info.cern.ch/hypertext/WWW/Addressing/Addressing.html** [8]

[11] N. Borenstein and N. Freed, MIME (Multipurpose Internet Mail Extensions): Mechanisms for Specifying and Describing the Format of Internet Message Bodies, Internet RFC 1341 (June 1992).

[12] A. Clausnitzer, Realization of WWW and ASCII Interfaces for the OMNIS Retrieval Component, Master Thesis, Technical University of Munich (November 1994).

[13] Conolly, D.: HyperText Markup Language, Internet Draft, July 1993.

[14] Frystyk H., Lie H.W.: Towards a Uniform Library of Common Code, *Second International WWW Conference*, Chicago, 1994.

[15] International Organization for Standardization: Information Processing -Text and Office Systems, Standard Generalized Markup Language (SGML), ISO 8879 1986 (Geneva, 1988).

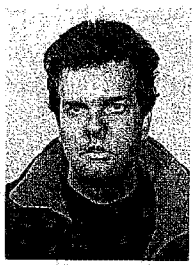
[16] Kantor P., Lapsley P.: A proposed standard for the transmission of news, Internet RFC 977.

[17] K. Meyer-Wegener, *Multimedia Databases* (Teubner Verlag, Stuttgart, 1991).

[18] NCSA, NCSA httpd (1995). **http://hoohoo.ncsa.uiuc.edu/docs/** [9]

[19] J. Postel and J. Reynolds, File Transfer Protocol, Internet RFC 959 (October 1985).

[20] TransAction Software GmbH, Myriad System and Administration Guide (Munich, 1993).



Alexander Clausnitzer is a Master student of Computer Science at Technische Universität in München. The topic of his Master Thesis was the "Realization of the WWW Version of the OMNIS Literature Retrieval". His scientific interests are World-Wide Web and multimedia systems.

[5] ftp://quake.think.com/pub/wais/wais-discussion/bibliography.txt
[6] http://info.cern.ch/hypertext/WWW/MarkUp/MarkUp.html
[7] http://info.cern.ch/hypertext/WWW/Protocols/HTTP/HTTP2.html
[8] http://info.cern.ch/hypertext/WWW/Addressing/Addressing.html
[9] http://hoohoo.ncsa.uiuc.edu/docs/

ORCL01806618



Pavel A. Vogel is employed as a scientist at the Department of Computer Science of Technische Universität in München. His interests include multimedia databases, digital libraries, and office information and library systems. His current work is concerned with the management and the development of the OMNIS library system. Vogel received his M.Sc degree in computer science from the Czech Technical University in Prague in 1965.



Stephan G. Wiesener is PhD student of computer science at FORWISS (Bavarian Research Center for Knowledge-Based Systems) in Munich. He manages digital library projects. His research interests include digital libraries, hypermedia systems, and knowledge-based systems. His current work concerns intelligent hypermedia systems. Wiesener received his M.Sc degree in computer science from Technische Universität in München in 1992.

CONFIDENTIAL TREATMENT REQUESTED

A13

THE AMERICAN HERITAGE® DICTIONARY

OF THE ENGLISH LANGUAGE

THIRD EDITION



HOUGHTON MIFFLIN COMPANY

Boston · New York

Words are included in this Dictionary on the basis of their usage. Words that are known to have current trademark registrations are shown with an initial capital and are also identified as trademarks. No investigation has been made of common-law trademark rights in any word, because such investigation is impracticable. The inclusion of any word in this Dictionary is not, however, an expression of the Publisher's opinion as to whether or not it is subject to proprietary rights. Indeed, no definition in this Dictionary is to be regarded as affecting the validity of any trademark.

American Heritage® and the eagle logo are registered trademarks of Forbes Inc. Their use is pursuant to a license agreement with Forbes Inc.

Houghton Mifflin Company gratefully acknowledges Mead Data Central, Inc., providers of the LEXIS®/NEXIS® services, for its assistance in the preparation of this edition of *The American Heritage® Dictionary.*

Copyright © 1996, 1992 by Houghton Mifflin Company.
All rights reserved.

No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, or by any information storage or retrieval system without the prior written permission of Houghton Mifflin Company unless such copying is expressly permitted by federal copyright law. Address inquiries to Reference Permissions, Houghton Mifflin Company, 222 Berkeley Street, Boston, MA 02116.

Library of Congress Cataloging-in-Publication Data

The American heritage dictionary of the English language.
–3rd ed.
p. cm.
ISBN 0-395-44895-6
1. English language–Dictionaries.
PE1628.A623 1992 92-851
423–dc20 CIP

Manufactured in the United States of America

For information about this and other Houghton Mifflin trade and reference books and multimedia products, visit The Bookstore at Houghton Mifflin on the World Wide Web at http://www.hmco.com/trade/.

to deprive of power : Latin *ex-*, ex- + Germanic **magan*, to be able to; see **magh-** in Appendix).] —**dis·may′ing·ly** *adv.*

SYNONYMS: *dismay, appall, daunt, horrify, shake.* These verbs mean to deprive a person of courage or the power to act as a result of fear or anxiety. *Dismay* is the least specific: *The news of plummeting stock prices dismayed speculators. Appall* implies a sense of helplessness caused by an awareness of the enormity of something: *"for as this appalling ocean surrounds the verdant land"* (Herman Melville). *Daunt* suggests an abatement of courage: *"captains courageous, whom death could not daunt"* (Anonymous ballad). *Horrify* implies dread, shock, or revulsion: *horrified by the possibility of nuclear war.* To *shake* is to dismay profoundly: *"A little swift brutality shook him to the very soul"* (John Galsworthy). See also Synonyms at **fear.**

dis·mem·ber (dĭs-mĕm′bər) *tr.v.* **-bered, -ber·ing, -bers.** **1.** To cut, tear, or pull off the limbs of. **2.** To divide into pieces. [Middle English *dismembren*, from Old French *desmembrer*, from Vulgar Latin **dismembrāre* : Latin *dis-*, dis- + Latin *membrum*, limb; see MEMBER.] —**dis·mem′ber·ment** *n.*

dis·miss (dĭs-mĭs′) *tr.v.* **-missed, -miss·ing, -miss·es.** **1.** To end the employment or service of; discharge. **2.** To direct or allow to leave: *dismissed troops after the inspection; dismissed the student after reprimanding him.* **3.a.** To stop considering; rid one's mind of; dispel: *dismissed all thoughts of running for office.* **b.** To refuse to accept or recognize; reject: *dismissed the claim as highly improbable.* **4.** *Law.* To put (a claim or action) out of court without further hearing. **5.** *Sports.* To put out (a batter) in cricket. [Middle English *dismissen*, from Medieval Latin *dismittere, dismiss-*, variant of Latin *dimittere* : *dī-, dis-*, apart; see DIS– + *mittere*, to send.] —**dis·miss′i·ble** *adj.* —**dis·mis′sion** (-mĭsh′ən) *n.*

SYNONYMS: *dismiss, boot, bounce, can, cashier, discharge, drop, fire, sack.* The central meaning shared by these verbs is "to terminate the employment of": *was dismissed for insubordination; was booted for being habitually tardy; afraid of being bounced for union activities; wasn't canned because his father-in-law owns the business; will be cashiered from the army; resort workers discharged at the end of the season; was dropped for incompetence; was fired on the spot for insolence; a reporter sacked for revealing a confidential source.* See also Synonyms at **eject.**

dis·miss·al (dĭs-mĭs′əl) *n.* **1.a.** The act of dismissing. **b.** The condition of being dismissed. **2.** An order or notice of discharge.

dis·mis·sive (dĭs-mĭs′ĭv) *adj.* **1.** Serving to dismiss. **2.** Showing indifference or disregard: *a dismissive shrug.*



Walt Disney

dis·mount (dĭs-mount′) *v.* **-mount·ed, -mount·ing, -mounts.** —*intr.* To get off or down, as from a horse or vehicle. —*tr.* **1.** To remove from a support, setting, or mounting. **2.** To unseat or throw off, as from a horse. **3.** To disassemble (a mechanism, for example). —**dismount** (dĭs′mount′) *n.* **1.** The act or manner of dismounting, especially from a horse. **2.** *Sports.* A move in gymnastics whereby the gymnast gets off an apparatus or completes a floor exercise, typically landing on both feet. [Probably alteration of obsolete French *desmonter*, to unseat : *des-*, dis- + *monter*, to mount; see MOUNT[1].] —**dis·mount′a·ble** *adj.*

Dis·ney (dĭz′nē), **Walter Elias.** Known as "Walt." 1901–1966. American animator, showman, and film producer. Noted for his creation of the cartoon characters Mickey Mouse and Donald Duck, he produced the first animated film with sound, *Steamboat Willie* (1928), and the first full-length animated feature, *Snow White* (1938).

dis·o·be·di·ence (dĭs′ə-bē′dē-əns) *n.* Refusal or failure to obey. —**dis′o·be′di·ent** *adj.* —**dis′o·be′di·ent·ly** *adv.*

dis·o·bey (dĭs′ə-bā′) *v.* **-beyed, -bey·ing, -beys.** —*intr.* To refuse or fail to follow an order or a rule. —*tr.* To refuse or fail to obey (an order or a rule). [Middle English *disobeien*, from Old French *desobeir*, from Vulgar Latin **disobedīre* : Latin *dis-*, dis- + Latin *oboedīre*, to obey; see OBEY.] —**dis′o·bey′er** *n.*

dis·o·blige (dĭs′ə-blīj′) *tr.v.* **-bliged, -blig·ing, -blig·es.** **1.** To refuse or neglect to act in accord with the wishes of. **2.** To inconvenience. **3.** To give offense to; affront. —**dis′o·blig′ing·ly** *adv.*

dis·or·der (dĭs-ôr′dər) *n.* **1.** A lack of order or regular arrangement; confusion. **2.** A breach of civic order or peace; a public disturbance. **3.** An ailment that affects the function of mind or body: *eating disorders and substance abuse.* —**disorder** *tr.v.* **-dered, -der·ing, -ders.** **1.** To throw into confusion or disarray. **2.** To disturb the normal physical or mental health of; derange.

dis·or·dered (dĭs-ôr′dərd) *adj.* **1.** Being in a condition of confusion or disarray. **2.** Physically or mentally ill. —**dis·or′dered·ly** *adv.* —**dis·or′dered·ness** *n.*

dis·or·der·ly (dĭs-ôr′dər-lē) *adj.* **1.** Lacking regular or logical order or arrangement: *a disorderly pile of clothes.* **2.** Undisciplined; unruly: *disorderly youths.* **3.** *Law.* Disturbing the public peace or decorum. —**dis·or′der·li·ness** *n.*

disorderly conduct *n. Law.* An offense involving disturbance of the public peace and decency.

dis·or·gan·ize (dĭs-ôr′gə-nīz′) *tr.v.* **-ized, -iz·ing, -iz·es.** To destroy the organization, systematic arrangement, or unity of. —**dis·or′gan·i·za′tion** (-gə-nī-zā′shən) *n.*

dis·o·ri·ent (dĭs-ôr′ē-ĕnt′, -ōr′-) *t*A13-03·ed, **-ent·ing, -ents.** To cause (a person, for example) to experience disorientation.

dis·o·ri·en·ta·tion (dĭs-ôr′rē-ĕn-tā′shən) *n.* **1.** Loss of one's sense of direction, position, or relationship with one's surroundings. **2.** Intellectual or moral confusion. **3.** *Psychology.* A temporary or permanent state of confusion regarding place, time, or personal identity.

dis·own (dĭs-ōn′) *tr.v.* **-owned, -own·ing, -owns.** To refuse to acknowledge or accept as one's own; repudiate.

disp. *abbr.* Dispensary.

dis·par·age (dĭ-spăr′ĭj) *tr.v.* **-aged, -ag·ing, -ag·es.** **1.** To speak of in a slighting way; belittle. See Synonyms at **decry.** **2.** To reduce in esteem or rank. [Middle English *disparagen*, to degrade, from Old French *desparager* : *des-*, dis- + *parage*, high birth (from *per*, peer; see PEER[2]).] —**dis·par′age·ment** *n.* —**dis·par′ag·er** *n.* —**dis·par′ag·ing·ly** *adv.*

dis·pa·rate (dĭs′pər-ĭt, dĭ-spăr′ĭt) *adj.* Fundamentally distinct or different in kind; entirely dissimilar. [Latin *disparātus*, past participle of *disparāre*, to separate : *dis-*, apart; see DIS– + *parāre*, to prepare; see **perə-**[1] in Appendix.] —**dis′pa·rate·ly** *adv.* —**dis′pa·rate·ness** *n.*

dis·par·i·ty (dĭ-spăr′ĭ-tē) *n., pl.* **-ties.** **1.** The condition or fact of being unequal, as in age, rank, or degree; difference: *"narrow the economic disparities among regions and industries"* (Courtenay Slater). **2.** Unlikeness; incongruity. [French *disparité*, from Old French *desparite*, from Late Latin *disparitās* : Latin *dis-*, dis- + Late Latin *paritās*, equality; see PARITY[1].]

dis·par·lure (dĭs′pär-lo͝or′) *n.* **1.** A pheromone, $C_{19}H_{38}O$, produced by female gypsy moths. **2.** A synthetic substance used to attract male gypsy moths to traps. [New Latin *(Porthetria) dispār*, gypsy moth (from *dispār* : *dis-*, dis- + *pār*, equal; see PAR) + LURE.]

dis·pas·sion (dĭs-păsh′ən) *n.* Freedom from passion, bias, or emotion; objectivity.

dis·pas·sion·ate (dĭs-păsh′ə-nĭt) *adj.* Devoid of or unaffected by passion, emotion, or bias. See Synonyms at **fair**[1]. —**dis·pas′sion·ate·ly** *adv.* —**dis·pas′sion·ate·ness** *n.*

dis·patch also **des·patch** (dĭ-spăch′) —*tr.v.* **-patched, -patch·ing, -patch·es.** **1.** To relegate to a specific destination or send on specific business. See Synonyms at **send**[1]. **2.a.** To complete, transact, or dispose of promptly. **b.** To eat up (food); finish off (a dish or meal). **3.** To put to death summarily. —*n.* **1.** The act of sending off, as to a specific destination. **2.** The act of putting to death. **3.** Speed in performance or movement. See Synonyms at **haste.** **4.** (*also* dĭs′păch′). **a.** A written message, particularly an official communication, sent with speed. **b.** An important message sent by a diplomat or an officer in the armed forces. **5.** (*also* dĭs′păch′). A news item sent to a news organization, as by a correspondent. **6.** An organization or a conveyance for delivering goods. [Spanish *despachar* or Italian *dispacciare*, both probably ultimately from Old Provençal *empachar*, to impede, from Vulgar Latin **impāctāre*, frequentative of Latin *impingere*, dash against. See IMPINGE.]

dis·patch·er (dĭs-păch′ər) *n.* One that dispatches, as: **a.** One that sends out trains, buses, trucks, or cars according to a schedule. **b.** *Computer Science.* A routine that controls the order in which input and output devices obtain access to the processing system.

dis·pel (dĭ-spĕl′) *tr.v.* **-pelled, -pel·ling, -pels.** **1.** To rid one's mind of: *managed to dispel my doubts.* **2.** To drive away or off by or as if by scattering. See Synonyms at **scatter.** [Middle English *dispellen*, from Latin *dispellere* : *dis-*, apart; see DIS– + *pellere*, to drive; see **pel-**[5] in Appendix.]

dis·pen·sa·ble (dĭ-spĕn′sə-bəl) *adj.* **1.** Not essential; unimportant: *dispensable items of personal property.* **2.** Capable of being dispensed, administered, or distributed: *dispensable drugs.* **3.** Subject to dispensation, as a vow or church law. —**dis·pen′sa·bil′i·ty, dis·pen′sa·ble·ness** *n.*

dis·pen·sa·ry (dĭ-spĕn′sə-rē) *n., pl.* **-ries.** *Abbr.* **disp.** **1.** An office in a hospital, school, or other institution from which medical supplies, preparations, and treatments are dispensed. **2.** A public institution that dispenses medicines or medical aid.

dis·pen·sa·tion (dĭs′pən-sā′shən, -pĕn-) *n.* **1.a.** The act of dispensing. **b.** Something dispensed. **c.** A specific arrangement or system by which something is dispensed. **2.** An exemption or a release from an obligation or a rule, granted by or as if by an authority. **3.a.** An exemption from a church law, a vow, or another similar obligation granted in a particular case by an ecclesiastical authority. **b.** The document containing this exemption. **4.** *Theology.* **a.** The divine ordering of worldly affairs. **b.** A religious system or code of commands considered to have been divinely revealed or appointed. —**dis′pen·sa′tion·al** *adj.*

dis·pen·sa·to·ry (dĭ-spĕn′sə-tôr′ē, -tōr′ē) *n., pl.* **-ries.** A book in which the contents, preparation, and uses of medicines are described; a pharmacopoeia.

dis·pense (dĭ-spĕns′) *v.* **-pensed, -pens·ing, -pens·es.** —*tr.* **1.** To deal out in parts or portions; distribute. See Synonyms at **distribute.** **2.** To prepare and give out (medicines). **3.** To administer (laws, for example). **4.** To exempt or release, as from a duty or religious obligation. —*intr.* To grant a dispensation or an exemption. —***phrasal verb.*** **dispense with.** **1.** To manage without; forgo. **2.** To get rid of; do away with: *a country that has dispensed with tariff barriers.* [Middle English *dispensen*, from Old French *dispenser*, from Latin *dispēnsāre*, to distribute, fre-

the program in which Mark was studying was designed to cover a great deal of material in a brief period.

in·ten·si·fi·er (ĭn-tĕn′sə-fī′ər) *n. Grammar.* See **intensive.**

in·ten·si·fy (ĭn-tĕn′sə-fī′) *v.* **-fied, -fy·ing, -fies.** *—tr.* **1.** To make intense or more intense: *The press has intensified its scrutiny of the candidate's background.* **2.** To increase the contrast of (a photographic image). *—intr.* To become intense or more intense: *The search intensified as dusk approached.* **—in·ten′si·fi·ca′tion** (-fĭ-kā′shən) *n.*

in·ten·sion (ĭn-tĕn′shən) *n.* **1.** The state or quality of being intense; intensity. **2.** The act of becoming intense or more intense; intensification. **3.** *Logic.* The sum of the attributes contained in a term. [Latin *intēnsiō, intēnsiōn-,* from *intēnsus,* stretched. See INTENSE.] **—in·ten′sion·al** *adj.*

in·ten·si·ty (ĭn-tĕn′sĭ-tē) *n., pl.* **-ties.** **1.** Exceptionally great concentration, power, or force. **2.** *Physics.* The amount or degree of strength of electricity, light, heat, or sound per unit area or volume. **3.** *Color.* **a.** The strength of a color, especially the degree to which it lacks its complementary color. **b.** See **saturation** (sense 5).

in·ten·sive (ĭn-tĕn′sĭv) *adj.* **1.** Of, relating to, or characterized by intensity: *intensive training.* See Usage Note at **intense.** **2.** *Grammar.* Tending to emphasize or intensify: *an intensive adverb.* **3.** Possessing or requiring to a high degree. Often used in combination: *research-intensive; labor-intensive.* **4.** Relating to or being a method especially of land cultivation intended to increase the productivity of a fixed area by means of an increase in capital and labor. **5.** *Physics.* Having the same value for any subdivision of a thermodynamic system: *intensive pressure.* **—intensive** *n. Grammar.* A linguistic element, such as the adverb *extremely* or *awfully,* that provides force or emphasis. Also called *intensifier.* **—in·ten′sive·ly** *adv.* **—in·ten′sive·ness** *n.*

intensive care *n.* Continuous and closely monitored health care that is provided to critically ill patients.

intensive care unit *n. Abbr.* **ICU** A specialized section of a hospital containing the equipment, medical and nursing staff, and monitoring devices necessary to provide intensive care.

in·tent (ĭn-tĕnt′) *n.* **1.** Something that is intended; an aim or a purpose. See Synonyms at **intention.** **2.** *Law.* The state of one's mind at the time one carries out an action. **3.** Meaning; purport. **—intent** *adj.* **1.** Firmly fixed; concentrated: *an intent gaze.* **2.** Having the attention applied; engrossed: *The students, intent upon their books, did not hear me enter the room.* **3.** Having the mind and will focused on a specific purpose: *was intent on leaving within the hour; are intent upon being recognized.* **—idiom. for (or to) all intents and purposes.** In every practical sense; practically: *To all intents and purposes the case is closed.* [Middle English *entent,* from Old French, from Medieval Latin *intentus,* from Latin, an extending, from *intentus,* attentive to, strained, from past participle of *intendere,* to direct attention. See INTEND.] **—in·tent′ly** *adv.* **—in·tent′ness** *n.*

in·ten·tion (ĭn-tĕn′shən) *n.* **1.** A course of action that one intends to follow. **2.a.** An aim that guides action; an objective. **b. intentions.** Purpose with respect to marriage: *honorable intentions.* **3.** *Philosophy.* A concept arising from directing the attention toward an object. **4.** *Medicine.* The process by which or the manner in which a wound heals. **5.** *Archaic.* Import; meaning. [Middle English *entencioun,* from Old French *intention,* from Latin *intentiō, intentiōn-,* from *intentus,* intent, from past participle of *intendere,* to direct attention. See INTEND.]

SYNONYMS: *intention, intent, purpose, goal, end, aim, object, objective.* These nouns refer to what one intends to do or achieve. *Intention* simply signifies a course of action that one proposes to follow: *It is not my intention to argue with you. Intent* more strongly implies deliberateness: *The executor tried to comply with the intent of the testator. Purpose* strengthens the idea of resolution or determination: *"His purpose was to discover how long these guests intended to stay"* (Joseph Conrad). *Goal* may suggest an idealistic or even a remote purpose: *"Black Power . . . is a call for black people to begin to define their own goals"* (Stokely Carmichael and Charles V. Hamilton). *End* suggests a long-range goal: *It has been said that the end justifies the means. Aim* stresses the direction one's efforts take in pursuit of an end: *The aim of every performing artist is to achieve perfection of execution.* An *object* is an end that one tries to carry out: *"The chief object of the English was to establish . . . a great empire on the Continent"* (Macaulay). *Objective* often implies that the end or goal can be reached: *"A major objective* [of political liberalism] *is the protection of the economic weak"* (Wayne Morse).

in·ten·tion·al (ĭn-tĕn′shə-nəl) *adj.* **1.** Done deliberately; intended: *an intentional slight.* See Synonyms at **voluntary.** **2.** Having to do with intention. **—in·ten′tion·al′i·ty** (-năl′ĭ-tē) *n.* **—in·ten′tion·al·ly** *adv.*

intentional community *n.* A small, localized, often rural community of persons or families pursuing common interests or concentrating on certain basic values.

in·ter (ĭn-tûr′) *tr.v.* **-terred, -ter·ring, -ters.** To place in a grave or tomb; bury. [Middle English *enteren,* from Old French *enterrer,* from Medieval Latin *interrāre* : Latin *in-,* in; see IN-[2] + Latin *terra,* earth; see **ters-** in Appendix.]

inter. *abbr.* Intermediate.

inter- *pref.* **1.** Between; among: *international.* **2.** In the midst of; within: *intertropical.* **3.** Mutual; mutually: *interrelate.* **4.** Reciprocal; reciprocally: *intermingle.* [Middle English *entre-, inter-,* from Old French *entre-,* from Latin *inter-,* from *inter,* between, among. See **en** in Appendix.]

in·ter·a·bang (ĭn-tĕr′ə-băng′) *n.* Variant of **interrobang.**

in·ter·act (ĭn′tər-ăkt′) *intr.v.* **-act·ed, -act·ing, -acts.** To act on each other: *"More than a dozen variable factors could interact, with their permutations running into the thousands"* (Tom Clancy).

in·ter·ac·tion (ĭn′tər-ăk′shən) *n.* **1.a.** The act or process of interacting. **b.** The state of undergoing interaction. **2.** *Physics.* Any of four fundamental ways in which elementary particles and bodies can influence each other, classified as strong, weak, electromagnetic, and gravitational.

in·ter·ac·tive (ĭn′tər-ăk′tĭv) *adj.* **1.** Acting or capable of acting on each other. **2.** *Computer Science.* Of or relating to a two-way electronic or communications system in which response is direct and continual. **3.** Of, relating to, or being a form of television entertainment in which the signal activates electronic apparatus in the viewer's home or the viewer uses the apparatus to affect events on the screen, or both. **—in′ter·ac′tive·ly** *adv.*

interactive terminal *n. Computer Science.* A computer or data-processing terminal capable of providing a two-way communication with the system to which it is connected.

in·ter·a·gen·cy (ĭn′tər-ā′jən-sē) *adj.* Involving or representing two or more agencies, especially government agencies.

in·ter a·li·a (ĭn′tər ā′lē-ə, ä′lē-ə) *adv.* Among other things. [Latin : *inter,* among + *alia,* neuter accusative pl. of *alius,* other.]

inter a·li·os (ā′lē-ōs′, ä′lē-ōs′) *adv.* Among other persons. [Latin : *inter,* among + *aliōs,* masculine accusative pl. of *alius.*]

in·ter·a·tom·ic (ĭn′tər-ə-tŏm′ĭk) *adj.* Occurring, operating, or situated between atoms.

in·ter·bank (ĭn′tər-băngk′) *adj.* Relating to, involving, or connecting two or more banks: *interbank borrowing; an interbank network of automated teller machines.*

in·ter·brain (ĭn′tər-brān′) *n.* See **diencephalon.**

in·ter·breed (ĭn′tər-brēd′) *v.* **-bred** (-brĕd′), **-breed·ing, -breeds.** *—intr.* **1.** To breed with another kind or species; hybridize. **2.** To breed within a narrow range or with closely related types or individuals; inbreed. *—tr.* To cause to interbreed.

in·ter·ca·lar·y (ĭn-tûr′kə-lĕr′ē, ĭn′tər-kăl′ə-rē) *adj.* **1.a.** Inserted in the calendar to make the calendar year correspond to the solar year. Used of a day or month. **b.** Having such a day or month inserted. Used of a year. **2.** Inserted between other elements or parts; interpolated. [Latin *intercalārius, intercalāris,* from *intercalāre,* to intercalate. See INTERCALATE.]

in·ter·ca·late (ĭn-tûr′kə-lāt′) *tr.v.* **-lat·ed, -lat·ing, -lates.** **1.** To insert (a day or month) in a calendar. **2.** To insert, interpose, or interpolate. [Latin *intercalāre, intercalāt-* : *inter-,* inter- + *calāre,* to proclaim; see **kelə-**[2] in Appendix.] **—in·ter′ca·la′tion** *n.* **—in·ter′ca·la′tive** *adj.*

in·ter·cede (ĭn′tər-sēd′) *intr.v.* **-ced·ed, -ced·ing, -cedes.** **1.** To plead on another's behalf. **2.** To act as mediator in a dispute. [Latin *intercēdere,* to intervene : *inter-,* inter- + *cēdere,* to go; see **ked-** in Appendix.] **—in′ter·ced′er** *n.*

in·ter·cel·lu·lar (ĭn′tər-sĕl′yə-lər) *adj. Biology.* Located among or between cells: *intercellular fluid.*

in·ter·cept (ĭn′tər-sĕpt′) *tr.v.* **-cept·ed, -cept·ing, -cepts.** **1.a.** To stop, deflect, or interrupt the progress or intended course of: *intercepted me with a message as I was leaving.* **b.** *Sports.* To take possession of by catching (an opponent's ball), especially in football. **2.** *Mathematics.* To include or bound (a part of a space or curve) between two points or lines. **3.** *Archaic.* To prevent. **4.** *Obsolete.* To cut off from access or communication. **—intercept** (ĭn′tər-sĕpt′) *n. Abbr.* **int.** **1.** *Mathematics.* The distance from the origin to the point at which a line, curve, or surface intersects a coordinate axis. **2.a.** The interception of a missile by another missile or an aircraft by another aircraft. **b.** Interception of a radio transmission. **3.** An interceptor. [Middle English *intercepten,* from Latin *intercipere, intercept-* : *inter-,* inter- + *capere,* to seize; see **kap-** in Appendix.] **—in′ter·cep′tive** *adj.*

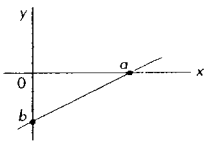
in·ter·cept·er (ĭn′tər-sĕp′tər) *n.* Variant of **interceptor.**

in·ter·cep·tion (ĭn′tər-sĕp′shən) *n.* **1.** The act of intercepting or the state of being intercepted. **2.** Something, such as a missile, an aircraft, or a radio transmission, that is intercepted. **3.** *Sports.* A pass that is intercepted, especially a forward pass in football.

in·ter·cep·tor also **in·ter·cept·er** (ĭn′tər-sĕp′tər) *n.* One that intercepts, specifically a fast-climbing, highly maneuverable fighter plane designed to intercept enemy aircraft or a guided missile designed to intercept enemy missiles and spacecraft.

in·ter·ces·sion (ĭn′tər-sĕsh′ən) *n.* **1.** Entreaty in favor of another, especially a prayer or petition to God in behalf of another. **2.** Mediation in a dispute. [Middle English, from Old French, from Latin *intercessiō, intercessiōn-,* intervention, from *intercessus,* past participle of *intercēdere,* to intervene. See INTERCEDE.] **—in′ter·ces′sion·al** *adj.* **—in′ter·ces′sor** (-sĕs′ər) *n.* **—in′ter·ces′so·ry** *adj.*

in·ter·change (ĭn′tər-chānj′) *v.* **-changed, -chang·ing, -chang·es.** *—tr.* **1.** To switch each of (two things) into the place of the other. **2.** To give and receive mutually; exchange. **3.** To cau[A13-04]ed each other in a series or pattern; alternate: *in-*



intercept
Intercept form of the equation of a line:
$\frac{x}{a}+\frac{y}{b}=1$

ă pat	oi boy
ā pay	ou out
âr care	o͝o took
ä father	o͞o boot
ĕ pet	ŭ cut
ē be	ûr urge
ĭ pit	th thin
ī pie	*th* this
îr pier	hw which
ŏ pot	zh vision
ō toe	ə about, item
ô paw	♦ regionalism

Stress marks: ′ (primary) ″ (secondary), as in **dictionary** (dĭk′shə-nĕr′ē)

Case 1:06-cv-00414-SLR Document 236-6 Filed 08/07/2008 Page 20 of 50

lifes, city scenes, and metaphorical works, such as *The Snake Charmer* (1907).

Rousseau, Jean Jacques. 1712–1778. French philosopher and writer who held that humanity is essentially good but corrupted by society. His written works include *The Social Contract* and the novel *Emile* (both 1762).

Rousseau, Théodore. 1812–1867. French landscape painter who was the leader of the Barbizon school. His works include *Descent of the Cattle* (c. 1834).

Rous·sil·lon (rōō-sē-yôn′). A historical region of southern France bordering on Spain and the Mediterranean Sea. Originally inhabited by Iberians, it became part of Roman Gaul after c. 121 B.C. and later changed hands many times, eventually becoming a Spanish possession that was transferred to France by the Treaty of the Pyrenees (1659).

roust (roust) *tr.v.* **roust·ed, roust·ing, rousts.** To rout, especially out of bed. [Probably alteration of ROUSE.]

roust·a·bout (rous′tə-bout′) *n.* **1.** A laborer employed for temporary or unskilled jobs, as in an oil field. **2.** A circus laborer. **3.** A deck or wharf laborer, especially on the Mississippi River.

rout[1] (rout) *n.* **1.a.** A disorderly retreat or flight following defeat. **b.** An overwhelming defeat. **2.a.** A disorderly crowd of people; a mob. **b.** People of the lowest class; rabble. **3.** A public disturbance; a riot. **4.** A company, as of knights or wolves, that are in movement. **5.** A fashionable gathering. —**rout** *tr.v.* **rout·ed, rout·ing, routs. 1.** To put to disorderly flight or retreat: *"the flock of starlings which Jasper had routed with his gun"* (Virginia Woolf). **2.** To defeat overwhelmingly. See Synonyms at **defeat.** [Middle English *route*, from Old French, troop, defeat, from Vulgar Latin **rupta*, from feminine of Latin *ruptus*, past participle of *rumpere*, to break. See **reup-** in Appendix.]

rout[2] (rout) *v.* **rout·ed, rout·ing, routs.** —*intr.* **1.** To dig with the snout; root. **2.** To poke around; rummage. —*tr.* **1.** To expose to view as if by digging; uncover. **2.** To hollow, scoop, or gouge out. **3.** To drive or force out as if by digging; eject: *rout out an informant.* **4.** *Archaic.* To dig up with the snout. [Variant of ROOT[2].]

rout[3] (rout, rōōt) *intr.v.* **rout·ed, rout·ing, routs.** *Chiefly British.* To bellow. Used of cattle. [Middle English *routen*, to roar, from Old Norse *rauta*.]

route (rōōt, rout) *n. Abbr.* **rte. 1.a.** A road, course, or way for travel from one place to another. **b.** A highway. **2.** A customary line of travel. See Synonyms at **way. 3.** A fixed course or territory assigned to a salesperson or delivery person. **4.** A means of reaching a goal. —**route** *tr.v.* **rout·ed, rout·ing, routes. 1.** To send or forward by a specific route. See Synonyms at **send**[1]. **2.** To schedule the order of (a sequence of procedures). [Middle English, from Old French, from Latin *rupta (via)*, broken (road), feminine past participle of *rumpere*, to break. See ROUT[1].]

rout·er[1] (rou′tər) *n.* One that routs, especially a machine tool that mills out the surface of metal or wood.

rout·er[2] (rōō′tər, rou′-) *n.* One that routes, especially one who prepares shipments for distribution and delivery.

rou·tine (rōō-tēn′) *n.* **1.** A prescribed, detailed course of action to be followed regularly; a standard procedure. **2.** A set of customary and often mechanically performed procedures or activities. See Synonyms at **method. 3.** A set piece of entertainment, especially in a nightclub or theater. **4.** *Slang.* A particular kind of behavior or activity: *She went into her hurt routine.* **5.** *Computer Science.* A set of programming instructions designed to perform a specific, limited task. —**routine** *adj.* **1.** In accord with established procedure: *a routine check of passports.* **2.** Habitual; regular: *made his routine trip to the store.* **3.** Having no special quality; ordinary: *a routine day.* [French, from *route*, route, from Old French. See ROUTE.] —**rou·tine′ly** *adv.* —**rou·tin′ism** *n.* —**rou·tin′ist** *n.*

rou·tin·ize (rōō-tē′nīz′, rōōt′n-īz′) *tr.v.* **-ized, -iz·ing, -iz·es. 1.** To establish a routine for. **2.** To reduce to a routine: *a government that routinized mass murder while carrying out its totalitarian policies.* —**rou·tin′i·za′tion** (-ĭ-zā′shən) *n.*

roux (rōō) *n., pl.* **roux.** A mixture of flour and fat cooked together and used as a thickening. [French *(beurre) roux*, browned (butter), from Old French *rous*, reddish brown, from Latin *russus*, red. See **reudh-** in Appendix.]

Rou·yn (rōō′ĭn, rwăn). A city of southwest Quebec, Canada, near the Ontario border west-northwest of Quebec City. It is a trade center in a mining region. Population, 17,224.

rove[1] (rōv) *v.* **roved, rov·ing, roves.** —*intr.* To wander about at random, especially over a wide area; roam. —*tr.* To roam or wander around, over, or through. See Synonyms at **wander.** —**rove** *n.* An act of wandering about, over, around, or through. [Middle English *roven*, to shoot arrows at a mark.]

rove[2] (rōv) *tr.v.* **roved, rov·ing, roves. 1.** To card (wool). **2.** To put (fibers) through an eye or opening. **3.** To stretch and twist (fibers) before spinning; ravel out. —**rove** *n.* A slightly twisted and extended fiber or sliver. [Origin unknown.]

rove[3] (rōv) *v. Nautical.* A past tense and a past participle of **reeve**[2].

rove beetle *n.* Any of numerous beetles of the family Staphylinidae, often found in decaying matter and having slender bodies and short wing covers. Also called *staphylinid.* [Possibly from ROVE[1].]

rov·er[1] (rō′vər) *n.* **1.a.** One that roves; a wanderer. **b.** A crewed or uncrewed vehicle, used especially in exploring the terrain of a planet and its satellites. **2.** *Sports.* A mark in archery selected by chance.

ro·ver[2] (rō′vər) *n.* **1.** A pirate. **2.** A pirate vessel. [Middle English, from Middle Dutch or Middle Low German, robber, from *roven*, to rob. See **reup-** in Appendix.]

Rov·no (rôv′nə). A city of northwest Ukraine west of Kiev. It was annexed by Russia in 1793 and by Poland in 1921. The city was occupied by Soviet troops in 1939. Population, 221,000.

Ro·vu·ma (rō-vōō′mə). See **Ruvuma.**

row[1] (rō) *n.* **1.** A series of objects placed next to each other, usually in a straight line. **2.** A succession without a break or gap in time: *won the title for three years in a row.* **3.** A continuous line of buildings along a street. —**row** *tr.v.* **rowed, row·ing, rows.** To place in a row. —***idiom.* a tough row to hoe.** *Informal.* A difficult situation to endure. [Middle English, from Old English *rāw*.]

row[2] (rō) *v.* **rowed, row·ing, rows.** —*intr. Nautical.* To propel a boat with or as if with oars. —*tr.* **1.** *Nautical.* **a.** To propel (a boat) with or as if with oars. **b.** To carry in or on a boat propelled by oars. **c.** To use (a specified number of oars or people deploying them). **2.** To propel or convey in a manner resembling rowing of a boat. **3.** *Sports.* **a.** To pull (an oar) as part of a racing crew. **b.** To race against by rowing. —**row** *n. Nautical.* **1.a.** The act of rowing. **b.** A shift at the oars of a boat. **2.** A trip or an excursion in a rowboat. [Middle English *rowen*, from Old English *rōwan*. See **erə-** in Appendix.] —**row′er** *n.*

row[3] (rou) *n.* **1.** A boisterous disturbance or quarrel; a brawl. See Synonyms at **brawl. 2.** An uproar; a great noise. —**row** *intr.v.* **rowed, row·ing, rows.** To take part in a quarrel, a brawl, or an uproar. [Origin unknown.]

row·an (rō′ən, rou′-) *n.* A small deciduous European tree *(Sorbus aucuparia)* of the rose family, having pinnately compound leaves, corymbs of white flowers, and orange-red berries. [Of Scandinavian origin. See **reudh-** in Appendix.]

row·boat (rō′bōt′) *n.* **1.** *Nautical.* A small boat propelled by oars. **2.** *Sports.* A rowing machine.

row·dy (rou′dē) *n., pl.* **-dies.** A rough, disorderly person. —**rowdy** *adj.* **-di·er, -di·est.** Disorderly; rough: *rowdy teenagers; a rowdy beer party.* [Probably from ROW[3].] —**row′di·ly** *adv.* —**row′di·ness** *n.* —**row′dy·ism** *n.*

Rowe (rō), **Nicholas.** 1674–1718. English writer whose works include drama, poetry, and an edition of Shakespeare. He was appointed poet laureate in 1715.

row·el (rou′əl) *n.* A sharp-toothed wheel inserted into the end of the shank of a spur. [Middle English, from Old French *roelle*, diminutive of *roue*, wheel, from Latin *rota*. See **ret-** in Appendix.] —**row′el** *v.*

♦**row·en** (rou′ən) *n. New England.* A second crop, as of hay, in a season. [Middle English *rowein*, from Anglo-Norman *rewain*, variant of Old French *regain* : *re-*, re- + *gaaignier*, to till; see GAIN[1].]

row house (rō) *n.* One of a series of identical houses situated side by side and joined by common walls.

row·ing machine (rō′ĭng) *n. Sports.* A fitness device with oarlike handles, pivoting footrests, and an adjustable sliding seat, used to provide a thorough workout of all major muscle groups in the body.

Row·land Heights (rō′lənd). A community of southern California, a suburb of the Los Angeles–Long Beach metropolitan area. Population, 28,252.

Row·land·son (rō′lənd-sən), **Thomas.** 1756–1827. British caricaturist and illustrator of works by Sterne, Goldsmith, and others.

row·lock (rō′lŏk′) *n. Chiefly British.* An oarlock.

Ro·xas y A·cu·ña (rō′häs ē ä-kōōn′yə, -yä), **Manuel.** 1892–1948. Philippine politician who was the first president of the Republic of the Philippines (1946–1948).

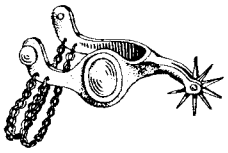
Roy (roi). A city of northern Utah, a suburb of Ogden. Population, 19,694.

roy·al (roi′əl) *adj. Abbr.* **R. 1.** Of or relating to a monarch. **2.** Of the rank of a monarch. **3.** Of, relating to, or in the service of a kingdom. **4.** Issued or performed by a monarch: *a royal warrant; a royal visit.* **5.** Founded, chartered, or authorized by a monarch: *a royal society of musicians.* **6.** Befitting royalty; stately: *royal treatment.* **7.a.** Superior, as in size or quality. **b.** Used as an intensive: *"It would be a first-class royal mess"* (Sam Nunn). —**royal** *n.* **1.** *Informal.* A member of a monarch's family: *"Among the resort's distinguished visitors are Swedish and Spanish royals"* (Alistair Scott). **2.** *Nautical.* A sail set on the royal-mast. **3.** A paper size, 20 by 25 inches for printing, 19 by 24 inches for writing. —***idiom.* the royal road.** A way or method that presents no difficulties: *the royal road to success.* [Middle English, from Old French, from Latin *rēgālis*, from *rēx, rēg-*, king. See **reg-** in Appendix.] —**roy′al·ly** *adv.*

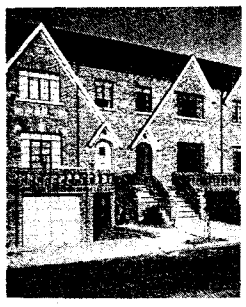
royal blue *n. Color.* A deep to strong blue. —**roy′al-blue′** (roi′əl-blōō′) *adj.*

royal fern *n.* A deep-rooted fern *(Osmunda regalis)* of worldwide distribution, having tall, upright, bipinnately compound fronds.

royal flush *n. Games.* A straight flush consisting of the five highest cards of one suit, ranked as the highest hand in certain games



rowel



row house

ă pat	oi boy
ā pay	ou out
âr care	o͝o took
ä father	o͞o boot
ĕ pet	ŭ cut
ē be	ûr urge
ĭ pit	th thin
ī pie	*th* this
îr pier	hw which
ŏ pot	zh vision
ō toe	ə about, item
ô paw	♦ regionalism

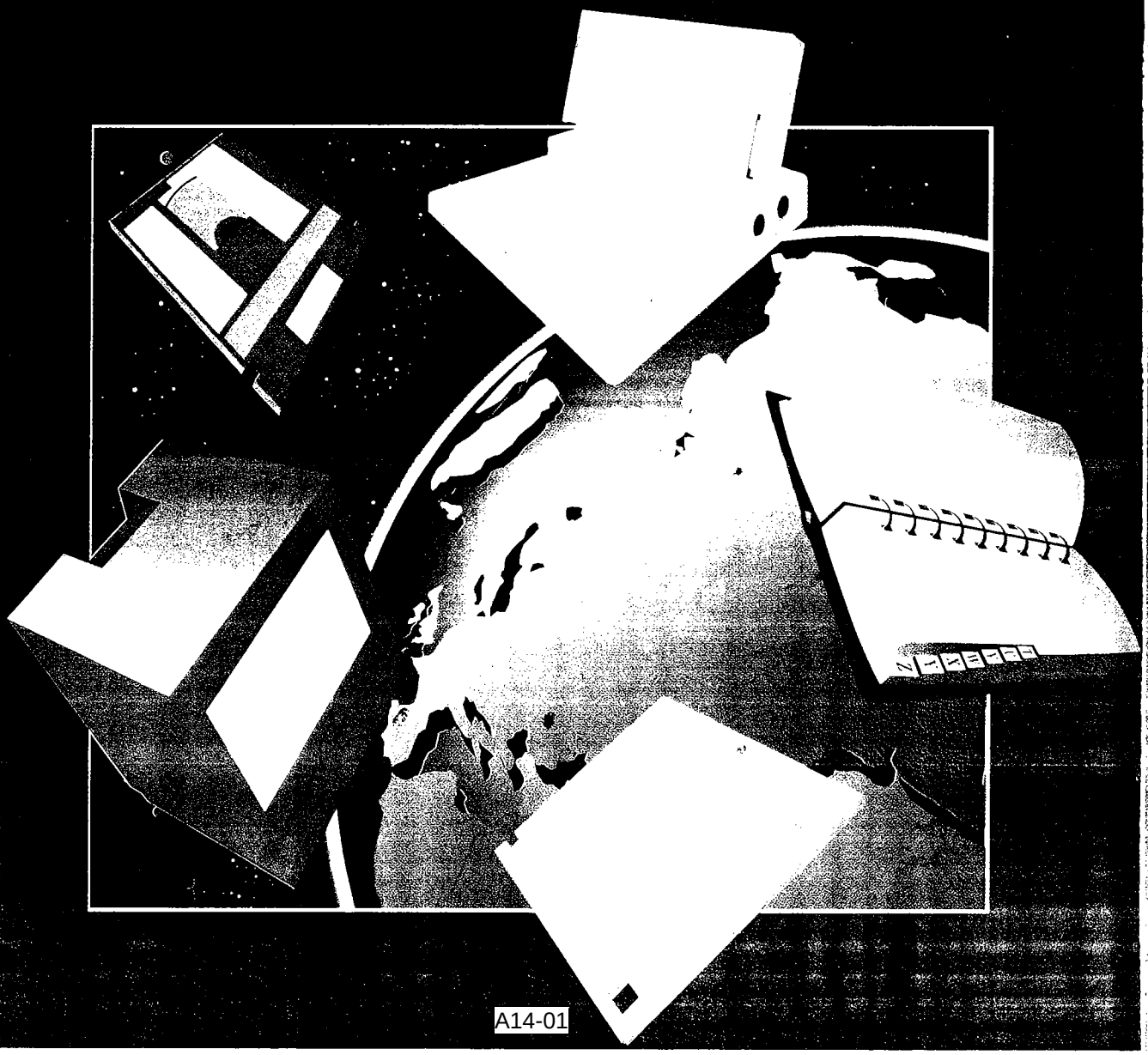
Stress marks: ′ (primary); ″ (secondary), as in **dictionary** (dĭk′shə-nĕr″ē)

A14

PETER COLLIN PUBLISHING
PROFESSIONAL SERIES

NEW

DICTIONARY OF PERSONAL COMPUTING AND THE INTERNET



DICTIONARY OF PERSONAL COMPUTING AND THE INTERNET

DICTIONARY OF
PERSONAL COMPUTING
AND THE
INTERNET

S.M.H.Collin

INDIANAPOLIS MARION CO. PUBLIC LIBRARY

PETER COLLIN PUBLISHING

First published in Great Britain 1997
published by
Peter Collin Publishing Ltd
1 Cambridge Road, Teddington, Middlesex, TW11 8DT

© Text Copyright Simon Colln 1997

All rights reserved. No part of this publication may be reproduced in any form or by any means without the permission of the publishers

British Library Cataloguing in Publication Data

A Catalogue record for this book is available from the British Library

ISBN 0-948549-93-9

Text computer typeset by PCP
Printed and bound in Great Britain by
Butler & Tanner

Cover design by Gary Weston

function in a Web page; for example, a WebBot can create a catalogue request form or can create a table for information. Instead of using complex HTML commands, you can enter the information into the WebBot and it will create the elements. If you want to use WebBots in your Web page, your Web server must support the WebBot extensions - check with your ISP

Web browser

see
BROWSER

Web crawler

software that moves over every new Web page on the internet and produces an index based on the content of the Web pages - normally used by search engines to ensure that they are up to date

Web page

single file stored on a Web server that contains formatted text, graphics and hypertext links to other pages on the internet. A Web page is created using HTML codes and is viewed with a browser

see
BROWSER, HTML, APPENDIX A

Web server

computer that stores the collection of Web pages that make up a Web site

what you see is what you get (WYSIWYG)

wordprocessing or DTP program where what you see on the screen is exactly the same as the image or text that will be printed, including graphics and special fonts

white pages

database of users and their email address stored on the internet to help other users find an email address; normally you have to add your own email address to the database

wide area information server

see
WAIS

wide area network

see
WAN

wild card character

symbol used when searching for files or data which represents all files; in DOS, UNIX and PC operating systems, the wild card character '?' will match any single character in this position; the wild card character 'or' means match any number of any characters

see also
QUESTION MARK

window

A15

GENERAL WEB TERMS - ORACLE'S PROPOSED CONSTRUCTIONS

Claim Term	Oracle's Proposed Construction	Support for Proposed Constructions
Connection Cache ('554 patent - Claim 4)	a store of information identifying open connections to data sources for eliminating subsequent connect times to those data sources	Col. 6:58-65[1] ("connection cache, containing connection to each of data source 406, data source 408 and data source 410, thus eliminating connect times … to those data sources.") "Connection Cache": a store of information identifying active connections.[2] A32 at p. 2.[3] **connection -** A physical link via wire, radio, fiber-optic cable, or other medium between two or more communications devices. *Microsoft Press Computer Dictionary* (3rd ed., 1997). A68 at p. 114. **connection** - (3)(B) (software) An association established between functional units for conveying information. *The IEEE Standard Dictionary of Electrical and Electronics Terms*, (6th ed., 1996). A69 at p. 203. **cache -** (2) A small portion of a high-speed memory used for temporary storage of frequently used

[1] All citations are to the '554 patent specification.

[2] EpicRealm stipulated to this construction in the Texas action.

[3] Appendix Exhibits are contained in Oracle's Appendix of Exhibits in Support of Oracle's Opening Claim Construction Brief and Motions Filed on July 31, 2008, and are subsequently referred to herein by number as "A___."

GENERAL WEB TERMS - ORACLE'S PROPOSED CONSTRUCTIONS

Claim Term	Oracle's Proposed Construction	Support for Proposed Constructions
		data, instructions, or operands. *The IEEE Standard Dictionary of Electrical and Electronics Terms* (6th ed., 1996). *Id.* at p. 124
Data Sources ('554 patent – claims 1-5, 8-9, 11 ('335 patent – claims [1], [15][4])	computerized repositories from which data may be retrieved electronically	Col. 5:43-48 ("Data sources, as used in the present application, include databases, spreadsheets, files and any other type of data repository.") "Data Sources": Databases, spreadsheets, files and any other type of data repository.[5] A32 at p. 2.
Logging Into ('554 patent – claim 5)	presenting credentials that allow access to	Col. 6:5-9; 6:38-42; 7:15-20 "Logging Into": Proving identification information to a data source.[6] A71 at p. 3. **login** - (1) The process of establishing communication with and verifying the authority to use a network or computer. *The IEEE Standard Dictionary of Electrical and Electronics Terms* (6th ed., 1996). A69 at p. 602. **log in** - (3) The act of identifying oneself as authorized to use a resource.

[4] Claim numbers in brackets signify unasserted independent claims upon which asserted dependent claims rely.

[5] EpicRealm stipulated to this construction in the Texas action.

[6] EpicRealm proposed this construction in its preliminary claim constructions in the Texas Action.

GENERAL WEB TERMS - ORACLE'S PROPOSED CONSTRUCTIONS

Claim Term	Oracle's Proposed Construction	Support for Proposed Constructions
		IBM Dictionary of Computing (1993 edition). A70 at p. 401.
Machine Readable Medium ('554 patent – claim 11)	a hard disk, floppy disk, CD-ROM, magnetic tape, or other magnetic or optical data storage medium	Col. 3:39-42 ("Data storage medium 108 may be a hard disk, a floppy disk, a CD-ROM, a magnetic tape, or other magnetic or optical data storage medium.")
Router ('554 patent – claims 9, 10)	a device that intercepts and dispatches dynamic Web page generation requests	'554 patent - claim 10 ("router includes … an interceptor intercepting said request … and a dispatcher … dispatching said request.") Col. 4:54-60 **router** - An intermediary device on a communications network that expedites message delivery. On a single network linking many computers through a mesh of possible connections, a router receives transmitted messages and forwards them to their correct destinations over the most efficient available route. On an interconnected set of local area networks (LANs) using the same communications protocols, a router serves the somewhat different function of acting as a link between LANs, enabling messages to be sent from one to another. *See also* bridge, gateway. *Microsoft Press Computer Dictionary* (3rd ed., 1997). A68 at p. 415.

61449312 v1

A16

REDACTED

IN ITS

ENTIRETY

A17

REDACTED

IN ITS

ENTIRETY

A18

EXHIBIT
WITHDRAWN

A19

REDACTED

IN ITS

ENTIRETY

A20

REDACTED

IN ITS

ENTIRETY

A21

REDACTED

IN ITS

ENTIRETY

A22

REDACTED

IN ITS

ENTIRETY

A23

REDACTED

IN ITS

ENTIRETY

A24

REDACTED

IN ITS

ENTIRETY

A25

REDACTED

IN ITS

ENTIRETY

A26

REDACTED

IN ITS

ENTIRETY

A27

REDACTED

IN ITS

ENTIRETY

A28

REDACTED

IN ITS

ENTIRETY

A29

EXHIBIT
WITHDRAWN

A30

REDACTED

IN ITS

ENTIRETY

A31

Resume of Michael Ian Shamos

ACADEMIC
Education
Foreign Languages
Honors
Editorships
Publications

LEGAL
Experience
Bar Admissions
Expert Witness
Legislative Testimony
Arbitration

ELECTRONIC VOTING
Experience
Legislative Testimony
Publications

BUSINESS
Experience
Consulting
Directorships

PERSONAL DATA
Contact Information

PUBLICATIONS
Science
Digital Libraries
Electronic Voting
Billiards
In Preparation

INVITED TALKS
Electronic Commerce
Science and Law
Electronic Voting

MISCELLANEOUS
Personal Data
Contact Information

Education

A.B. (1968) Princeton University (Physics). Thesis: "An Absorber Theory of Gravitational Radiation". Advisor: John A. Wheeler.

M.A. (1970) Vassar College (Physics). Thesis: "An Absorber Theory of Acoustical Radiation." Advisor: Morton A. Tavel.

M.S. (1972) American University (Technology of Management).

M.S. (1973) Yale University (Computer Science).

M.Phil. (1974) Yale University (Computer Science).

Ph.D. (1978) Yale University (Computer Science). Thesis: "Computational Geometry". Thesis committee: David Dobkin, Martin H. Schultz, Stanley C. Eisenstat.

J.D. (1981) Duquesne University, *cum laude*.

Foreign Languages

French, Russian (good reading and technical translation skills, fair conversational ability).

Academic Experience

Distinguished Career Professor, Institute for Software Research and Language Technologies Institute, School of Computer Science, Carnegie Mellon University (2001-). Principal Systems Scientist (1998-2001). Principal Lecturer (2002-2003). Teaching Professor (2003-). Faculty, Tepper School of Business, Carnegie Mellon University (1999-2004).

Co-Director, Carnegie Mellon Institute for eCommerce (1998-). Vice-Chair, University Research Council (2000-2002), Director, eBusiness Technology degree program (2003-).

Director, Universal Library, Carnegie Mellon University (1998-).

Visiting Professor, Department of Electrical and Electronic Engineering, The University of Hong Kong (2001-).

Adjunct Faculty, Carnegie Mellon University, Department of Computer Science (1981-1998). Formerly Assistant Professor, Carnegie Mellon University, Departments of Computer Science and Mathematics (1975-81), Dept. of Statistics (1978-81).

Recent courses taught (Carnegie Mellon):
Algorithm Design and Analysis 15-451 (Comp. Sci.)
Intellectual Capital and its Protection 45-886 (MBA)
Ecommerce Technology 20-751 (MSEC program)
Electronic Payment Systems 20-753 (MSEC program), 96-774 (MSIT program)
Ecommerce Law and Regulation 46-840 (MSEC program)
Electronic Voting 17-803
Ubiquitous Computing, 96-761

Honors and Awards

Fellow, Society of the Sigma Xi (1974-83).

IBM Fellowship, Yale University (1974–75).

SIAM National Lecturer (1977–78).

Distinguished Lecturer (computer science), University of Rochester (1978); McGill University (1979).

Duquesne University Law Review (1980–81).

Black & White Scotch Achiever's Award (first annual, 1991, for contributions to bagpipe musicography).

Industry Service Award of the Billiard and Bowling Institute of America, 1996 (for contributions to billiard history).

Billiard Worldcup Association official referee (2001-)

Editorships

Editor-in-Chief, Journal of Privacy Technology (2003- 2006).

Member of Editorial Board, *Electronic Commerce Research Journal* (2000-).

Member of Editorial Board, *Pittsburgh Journal of Technology, Law and Policy* (1999-).

Dr. Shamos has reviewed scientific papers for *Communications of the ACM*, *Mathematical Reviews*, *IEEE Computer*, *IEEE Transactions on Computers*, *Information Processing Letters*, *Journal of the ACM* and the *Journal of Computational Physics*.

Contributing Editor, *Billiards Digest* magazine (1990-).

Legal Experience

Special Counsel, Reed Smith LLP (2000-2003), electronic commerce law.

Shareholder, The Webb Law Firm (1996-2000), intellectual property law. Associate (1990-95).

Private practice of law (1987-90), intellectual property

Associate, law firm of Buchanan, Ingersoll, P. C. (1985-87), Emerging Companies Department.

General Counsel, Carnegie Group, Inc. (1983-85), artificial intelligence company.

Private practice of law (1981-83), computer law.

Bar Admissions

Supreme Court of Pennsylvania (1981–).

United States District Court for the Western District of Pennsylvania (1981–).

United States Patent and Trademark Office (1981–).

United States Tax Court (1982–).

United States Court of Appeals for the Armed Forces (1982–).

United States Court of Appeals for the Third Circuit (1982–).

United States Supreme Court (1985–).

United States Court of Appeals for the Federal Circuit (1985–).

Expert Witness

Dr. Shamos serves as an expert witness in computer software and electronic voting cases. He has participated in the following:

C.W. Communications, Inc. v. International Research Service, Inc., Civil Action No. 84-890, (W.D. Pa. 1984), aff'd. Case No. 88-3331 (3d Cir., Oct. 31, 1988). Dr. Shamos testified for plaintiff publisher as to the fame of its "Computerworld" trademark. Result: permanent injunction against defendant. Judge McCune's Memorandum and Order states. "We accept the conclusion drawn by Dr. Shamos."

E.F. Hutton, Inc. v. Gipson (W.D. Pa. 1985). Dr. Shamos testified for defendant-counterclaimant physician as to fraud in the inducement by a computer hardware supplier. Plaintiff had provided capital financing for the purchase. Result: defendant was awarded compensatory damages + $250,000 punitive damages.

In re Comprehensive Business Systems, 119 B.R. 573 (S.D. Ohio 1990). Dr. Shamos testified for a secured creditor in a bankruptcy case in which the creditor sought to obtain software still in development for which it had advanced over $2 million in funding. Dr. Shamos opined as to the value of the incomplete software. Result: the creditor was able to purchase the software from the Trustee for $67,500. The Court referred in its opinion to "the testimony of the eminent and impressive Dr. Shamos."

Levinson Steel Co. v. American Software, Inc. et al., Civil Action No. 96-282, W.D. Pa. (1996). Dr. Shamos testified for plaintiff concerning bad faith estimates of computer processing capacity resulting in delivery of an inadequate system. Result: settlement in favor of plaintiff in an undisclosed amount. Contact: Reed Smith LLP, 435 Sixth Ave., Pittsburgh, PA 15219.

ASE Limited v. INCO Alloys International, Inc., Civil Action No. 98-1266, (W.D. Pa. 1998). Dr. Shamos testified for defendant concerning breach of computer services contract by declaratory judgment plaintiff. Result: determination that defendant was free to seek services from a different vendor.

Twentieth Century Fox Film Corp. v. iCraveTV., 53 U.S.P.Q. 2d 1831 (W.D. Pa. 2000). Testified for Plaintiffs concerning Internet technology used to stream video from U.S. TV stations through web sites in Canada. Result: TRO and preliminary injunction issued against defendants prohibiting continued infringement in the U.S. Contact: Gregory Jordan, Esq., Reed Smith LLP, 435 Sixth Ave., Pittsburgh, PA 15219

Invited testimony before the British House of Lords, Subcommittee B of the European Union Committee, April 20, 2000. Subject: European regulation of eCommerce. View testimony.

Universal Studios, Inc. v. Reimerdes, 111 F. Supp. 2d 294 (S.D.N.Y. 2000), aff'd 273 F.3d 429 (2d Cir. 2001). Testified for plaintiff movie studios concerning accused software for decrypting DVDs in the first case interpreting the Digital Millennium Copyright Act. Result: permanent injunction issued in favor of plaintiffs on August 17, 2000. Contact: William Hart, Esq., Proskauer Rose LLP. View testimony. View opinion. View appellate opinion.

MercExchange, L.L.C. v. eBay, Inc. et al., Case No. 2:01-CV-736 (E.D. Va. 2001). Served as an expert for defendant eBay in an infringement case concerning U.S. Patent 6,202,051 for Internet auctions. Following Dr. Shamos' reports, Defendants obtained a summary judgment of noninfringement of the subject patent. The case continues on two other patents. Contact: Tim Teter, Esq., Cooley Godward Kronish LLP.

Powerquest Corp. v. Quarterdeck Corp. et al., Case No. 2:97-CV-0783 (D. Utah 1997). Served as an expert for plaintiff PowerQuest in an infringement case concerning U. S. Patents 5,675,769 and 5,706,472 for a method of resizing hard disk partitions. Dr. Shamos testified at the Markman hearing. Case settled when one of the defendants acquired plaintiff. Contact: Merchant & Gould,

Denver, CO.

Sightsound.Com Inc. v. N2K Inc. et al., C.A. 98-118 (W.D. Pa. 1998). Served as an expert for defendants, including a subsidiary of Bertelsmann AG, concerning validity of U.S. Patents 5,191,573 and 5,966,440 for distribution of digital audio via telecommunications lines. Case settled. Contact: Steven M. Hayes, Esq., Parcher, Hayes & Snyder, 500 Fifth Avenue, New York, NY 10110.

Freemarkets, Inc. v. B2eMarkets, Inc., C.A. 02-162-SLR (D. Del. 2002). Served as an expert witness for plaintiff concerning infringement of U.S. patents 6,216,114 and 6,223,167, concerning methods of conducting electronic auctions. Case settled two weeks after expert attended a demonstration of the accused product. Contact: Morgan, Lewis & Bockius LLP, 1111 Pennsylvania Ave. N.W., Washington, DC 20004.

Lifecast.com, Inc. v. ClubCorp, Inc., AAA Case No. 71Y1170076301 (Dallas, TX). Served as an expert witness for respondent in a case alleging copyright infringement of Internet websites. Testified at arbitration. Result: Complainant's claims denied; award for respondent on counterclaims and for attorney's fees. Contact: Bill Whitehill, Esq., Gardere Wynne Sewell LLP, 1601 Elm St., Dallas, TX 75201.

IP Innovation LLC v. Thomson Learning, Inc. et al., Case H-02-2031 (S.D. Tex. 2002). Served as a expert for defendant The Princeton Review, Inc. concerning alleged infringement of U.S. Patent 4,877,404 relating to online delivery of educational courses. Summary judgment of non-infringement obtained for defendant after favorable Markman proceeding. Contact: Peter Vogel, Esq., Gardere Wynne Sewell LLP, 1601 Elm St., Dallas, TX 75201.

Starpay.com LLC et al. v. Visa International Service Association et al., Case 3-03-CV-976-L (N.D. Tex. 2003). Served as an expert for defendant Visa concerning alleged infringement of U.S. Patent 5,903,878 relating to online authentication of credit card customers. Dr. Shamos provided the court with a Markman tutorial in 2004 and a non-infringement and invalidity declaration in 2008. Case settled in February 2008. Contact: Stanley Young, Esq., Heller Ehrman, LLP, 275 Middlefield Road, Menlo Park, CA

94025.

Safeclick LLC v. Visa International Service Association et al., Case C-03-5865 (N.D. Cal. 2003). Served as an expert for defendant Visa concerning alleged infringement of U.S. Patent 5,793,028 relating to online authentication of credit card customers. Summary judgment of noninfringement granted for Visa based on expert reports, affirmed after appeal to the Federal Circuit. Contact: Stanley Young, Esq., Heller Ehrman, LLP, 275 Middlefield Road, Menlo Park, CA 94025.

Wells Fargo Bank Minnesota, NA et al. v. UBS Warburg Real Estate Securities, Inc., Case 02-2849 (192d Judicial District, Dallas Cty., Tex, 2002) and *LaSalle Bank, NA et al. v. UBS Warburg Real Estate Securities, Inc.*, Case 02-2899-G (134th Judicial District, Dallas Cty., Tex, 2002). Served as an expert for defendant UBS Warburg in an electronic discovery matter involving a case of first impression regarding Texas Discovery Rule 196.4 allocating costs of discovery of electronic records. Status: pending. Contact: Dawn Estes, Esq., Gardere Wynne Sewell LLP, 1601 Elm St., Dallas, TX 75201.

American Association of People with Disabilities et al. v. Shelley et al., Case No. CV04-1526 FMC (PJWx) (C. D. Calif., 2004). Served as an expert for plaintiff AAPD, which has brought a claim against the California Secretary of State that requiring DRE voting machines to be equipped with audit trails violates the rights of disabled persons. Plaintiffs' application for TRO and preliminary injunction denied. Contact: John McDermott. Esq., Howrey Simon Arnold & White, 550 South Hope St., Suite 1100, Los Angeles, CA 90071.

Paul Ware v. Target Corp., CA 4:03-CV-0243-HLM (N.D. Ga., 2003). Served as an expert for defendant Target Corp., a large retailer, in a case involving U.S. patent 4,707,492, claiming a method of conducting credit card sales. Case settled during Markman preparations. Contact: Thomas Burke, Esq., Ropes & Gray LLP, 1211 Ave. of the Americas, New York, NY 10036.

Viad Corp., v. C. Alan Cordial et al., No. 03-1408 (W.D. Pa., filed 2003). Served as an expert for defendants in an action alleging misappropriation of

trade secrets relating to software for automating certain aspects of the exhibit booth and trade show industries. Status: case settled immediately before trial, after plaintiff's unsuccessful Daubert challenge of Dr. Shamos. Contact: Barbara Scheib, Esq., Cohen & Grigsby, P.C., 11 Stanwix Street, Pittsburgh, PA 15222.

Schade et al. v. Maryland State Bd. of Elections et al., Case No. C0497297 (Cir. Ct. Anne Arundel Cty. Md., 2004). Serving as an expert for defendants in a case challenging the decision of the Board of Elections not to decertify Diebold AccuVote system. Result: Plaintiff's motion for preliminary injunction denied, upheld on appeal. Judge Manck's opinion cites Dr. Shamos' testimony as follows: "the court finds Dr. Shamos, Defendants' expert, to be the true voice of reason and the most credible expert in this matter." The denial of preliminary injunction was upheld by the Maryland Court of Appeals, which commented extensively on Dr. Shamos' testimony in its opinion. Remainder of case is pending. Contact: Michael Berman, Esq., Deputy Attorney General, 200 St. Paul Place, Baltimore, MD 21202-2021.

Wexler et al. v. Lepore et al., Case No. 04-80216 (CIV-COHN) (S.D. Fla. 2004) . Served as an expert for defendants, various Florida election supervisors against claim by U.S. Congressman Robert Wexler that use of DRE voting machines without paper audit trails violates the equal protection clause of the U.S. Constitution. Dr. Shamos testified on Oct. 19, 2004. The trial judge rendered judgment in favor of defendants on Oct. 25. Contact: Jason Vail, Esq., Assistant Attorney General, Department of Legal Affairs, The Capitol, Tallahassee, FL 32399-1050. Opinion.

Siemens Information and Communication Networks, Inc. v. Inter-Commercial Business Systems, Inc., Civil Action 3-03CV2171-L (N.D. Tex. 2004). Served as an expert for defendant against claim of copyright infringement based on reverse-engineered firmware resident in telephone switching systems. Status: case settled shortly after the submission of Dr. Shamos's rebuttal report on non-infringement. Contact: Bill Whitehill, Esq., Gardere Wynne Sewell LLP, 1601 Elm St., Dallas, TX 75201.

Soverain Software LLC v. Amazon.com, Inc., C.A. No.

6:04-CV-14 (E.D. Tex. 2004). Served as an expert for plaintiff regarding asserted patents 5,708,780, 5,715,314 and 5,909,492, relating to the shopping cart paradigm of electronic commerce. Status: settled in Sept. 2005 with Amazon paying $40 million to Soverain and taking a license under the patents in suit. Contact: Thomas L. Giannetti, Esq., Jones Day, 222 E. 41st St., New York, NY 10017.

CollegeNET, Inc. v. The Princeton Review, Inc., Case '051205KI (D. Ore. 2005). Served as a expert for defendant The Princeton Review, Inc. concerning alleged infringement of U.S. Patent 6,460,042 relating to online delivery of educational courses. Case settled in December 2007. Contact: Peter Vogel, Esq., Gardere Wynne Sewell LLP, 1601 Elm St., Dallas, TX 75201.

CombineNet, Inc. v. Verticalnet. Inc., GD 05-018911 (Ct. Common Pleas, Allegheny Cty., PA). Served as an expert for plaintiff in an action for trade secret misappropriation relating to a system for conducting electronic auctions. Plaintiff won in arbitration. Contact: Mark Knedeisen, Esq., Kirkpatrick & Lockhart Preston Gates Ellis LLP, 535 Smithfield Street, Pittsburgh, PA 15222-2312.

RealSource, Inc. v. Best Buy Co., Inc. et al., No. A04-CA-771-LY (W.D. Tex.). Served as an expert for defendant Lowe's Companies, Inc., against a claim of infringement of U.S. patent 5,732,136 relating to validation of point-of-sale debit card transactions. Provided a tutorial to the Court during Markman proceedings concerning debit card technology. Defendants won summary judgment of non-infringement, now on appeal to the Federal Circuit. Defendants Lowe's has settled and is not involved in the appeal. Contact: Michael S. Connor, Esq., Alston & Bird LLP, Bank of America Plaza, 101 South Tryon St, Suite 4000, Charlotte, NC 28280-4000.

DE Technologies, Inc. v. Dell, Inc. et al., No. 7:04-CV-00628 (W.D. Va.). Served as an expert for plaintiff DE Technologies, Inc., asserting a claim of infringement of U.S. patents 6,460,020 and 6,845,364, relating to a system for implementing international sales transactions. Case settled after and adverse summary judgment. However, the Court used Dr. Shamos' testimony in its opinion. Contact: David Marder, Esq., Robins Kaplan Miller & Ciresi

LLP, 800 Boylston Street, 25th Floor, Boston, MA 02199.

Eaton Power Quality Corp. v. J.T. Packard & Associates, No. 05 C 3545 (N.D. Ill. 2005). Served as expert for plaintiff in a claim of software copyright infringement involving a system for configuring industrial uninterruptible power supplies. Case settled in early 2007. Contact: Keith Schoeneberger, Esq., LeBoeuf, Lamb, Greene & MacRae LLP, Suite 1175, 180 North Stetson Ave., Chicago, IL 60601-6783.

Taylor et al. v. Onorato et al., CA 06-481 (W.D. Pa 2006). Served as an expert for Commonwealth of Pennsylvania defendants in an action seeking to enjoin the use of electronic voting machines in Allegheny County, PA. Dr. Shamos testified at length in a preliminary injunction hearing held April 25-27, 2006. The injunction was denied on April 28. Suit was subsequently dropped by plaintiffs. Contact: Mark Aronchick, Esq., Hangley Aronchick Segal & Pudlin, One Logan Square, 18th & Cherry Streets, 27th Floor, Philadelphia, PA 19103.

FedEx Ground Package System, Inc. v. Applications International Corp., CA No. 03-1512 (W.D. Pa.). Serving as an expert for defendant counterclaiming for copyright infringement and trade secret misappropriation relating to software for maintaining occupational health and safety records. Case is pending. Contact: Ronald Hicks, Esq., Meyer, Unkovic & Scott LLP, 1300 Oliver Bldg., Pittsburgh, PA 15222.

NetMoneyIN, Inc. v. Verisign, Inc. et al., Cv-01-441-TUC-RCC (D. Ariz.). Served as an expert for defendants Bank of America Merchant Services, Inc. and Wells Fargo Bank, N.A., who are accused of infringing claim 23 of U.S. patent 5,822,737, relating to an electronic payment system. Wells Fargo and Bank of America have settled. Contact: Robert Smith, Esq., Kirkpatrick & Lockhart Preston Gates Ellis LLP, State Street Financial Center, One Lincoln Street, Boston, Massachusetts 02111-2950.

Contois Music Technology, LLC v. Apple Computer, Inc., 2:05-CV-163 (D. Vermont, filed Feb. 13, 2006). Served as an expert for plaintiff in an action alleging that the Apple iTunes software infringed U.S. patent

5,864,868, relating to a method for selecting music from am electronic catalog. Case settled after a favorable Markman order. Contact: John Rabena, Esq., Sughrue Mion, PLLC, 2100 Pennsylvania Avenue, NW, Suite 800, Washington, DC 20037-3213.

Banfield et al. v. Cortes, 442 MD 2006 (PA Cmwlth. Ct.). Serving as an expert for defendant Secretary of the Commonwealth of Pennsylvania in an action to compel the decertification of all electronic voting machines in Pennsylvania. Case is pending. In February 2008 Defendant successfully repelled an emergency motion for preliminary injunction. Contact: Alan Promer, Esq., Hangley Aronchick Segal & Pudlin, One Logan Square, 18th & Cherry Streets, 27th Floor, Philadelphia, PA 19103.

Remote Inventory Systems, Inc. v. WESCO Distribution, Inc., AAA Case No. 55 171 00493 05 (Pittsburgh, PA). Serving as an expert for respondent in a case alleging misappropriation of trade secrets in a computerized inventory system. Case is pending. Contact: Kirsten Rydstrom, Esq., Reed Smith LLP, 435 Sixth Ave., Pittsburgh, PA 15219.

SyncSort, Inc. v. Innovative Routines International, Inc., Civil Action No. 04-3623 (WHW) (D. New Jersey). Serving as an expert witness for defendant in an action alleging misappropriation of trade secrets embodied in plaintiff's Unix sorting software. Case is pending. Contact: David R. Fine, Esq., Kirkpatrick & Lockhart Preston Gates Ellis LLP, 17 N. Second Street, 18th Floor, Harrisburg, PA 17101-1507.

Digital Impact, Inc. v. Bigfoot Interactive, Inc., Civil Action C05 00636 (CW) (N.D. Cal.). Serving as an expert witness for defendant in an action alleging infringement of U.S. Patent 6,449,634, relating to determining which file formats can be processed by an email client. Case is pending. Contact: Arthur Dresner, Esq., Reed Smith LLP, 599 Lexington Avenue, New York, NY 10022.

Prism Technologies LLC v. Verisign, Inc. et al., CA 05-214-JJF (D. Del.). Serving as an expert for plaintiff in an action alleging infringement of U.S. Patent 6,516,416, relating to use of a hardware key for authentication over networks. Case is on appeal to the Federal Circuit. Contact: Robins Kaplan Miller &

Ciresi LLP.

AdvanceMe, Inc. v. Rapidpay LLC et al., Civil Action 6:05-cv-424 (E.D. Tex., Tyler Division). Served as an expert witness for plaintiff in an action alleging infringement of U.S. Patent 6,941,281, relating to an automated payment system for dividing credit card proceeds between a merchant and another party. Testified at a bench trial in July 2007 before Judge Davis, who held the patent invalid for obviousness. Case is on appeal to the Federal Circuit. Contact: Ronald S. Lemieux, Esq., Paul, Hastings, Janofsky & Walker LLP, Five Palo Alto Sq., Palo Alto, CA 94306.

IBM Corp. v. Amazon.com, Inc., CA 9:06-CV-242-RHC (E.D. Tex., Lufkin Div.) and *IBM Corp. v. Amazon.com, Inc.*, CA 6:06-CV-452-LED (E.D. Tex., Marshall Div.). Served as an expert for IBM in related actions alleging infringement of U.S. Patents 5,319,542, 5,442,771, 5,446,891, 5,796,967 and 7,072,849, all concerning methods of conducting electronic transactions, and a counterclaim for infringement of U.S. Patent 5,826,258, concerning a method for querying semistructured data. Case settled early in discovery when the parties cross-licensed each other's patents. Contact: Mark Ziegelbein, Esq., Jones Day, 2727 North Harwood Street, Dallas, TX 75201-1515.

The MathWorks, Inc. v. COMSOL AB et al., CA 6:06-CV-334 (E.D. Tex., Tyler Division). Serving as an expert for plaintiff The MathWorks, providers of the mathematical software system MATLAB, in an action alleging infringement of U.S. Patents 7,051,338 and 7,181,745 concerning methods for invoking object methods from external environments. Case is pending. Contact: Krista Schwartz, Esq., Jones Day, 77 W. Wacker Dr., Chicago, IL 60601-1692.

Avante Int'l. Technology Corp. v. Diebold Election Systems et al., Case 4:06-CV-0978 (E.D. Mo., Eastern Division). Serving as an expert for defendant Sequoia Voting Systems in an action alleging infringement of U.S. Patents 6,892,944, 7,036,730 and 7,077,313 concerning electronic voting technology. Case is pending. Contact: Peter Ewald, Esq., Oliff & Berridge, PLC, 277 South Washington Street, Suite 500, Alexandria, VA 22314.

Netcraft Corp. v. eBay, Inc. and PayPal, Inc., Case

3:07-cv:00254-bbc (W.D. Wisc. 2007). Served as an expert for defendants in an action alleging infringement of U.S. Patents 6,351,739 and 6,976,008 concerning methods of billing for ecommerce transactions over the Internet. Defendants were granted summary judgment of non-infringement on Dec, 10, 2007. Contact: Kenneth Weatherwax, Esq., Irell & Manella LLP, 180 Avenue of the Stars, Los Angeles, CA 90067.

ACLU of Ohio et al. v. Brunner et al., Case 1:09 CV 0145 (N.D. Ohio 2008). Served as an expert witness in an action alleging that the use of central count optical scan voting should not be permitted in Cuyahoga County, as had been ordered by the county board of elections. A preliminary injunction was denied in February 2008 and trial is set for July 2008. Contact: Meredith Bell-Platts, ACLU Voting Rights Project, 2600 Marquis One Tower, Atlanta, GA 30303.

Legislative Testimony

Testimony before the Texas Legislature concerning electronic voting, Austin, Texas, 1987. Result: passage of the Texas Electronic Voting Law.

Invited testimony before the British House of Lords, Subcommittee B of the European Union Committee, April 20, 2000. Subject: European regulation of eCommerce.

Testimony before the Pennsylvania Legislature State Government Committee concerning electronic voting, Philadelphia, March 10, 2004.

Testimony before the United States Commission on Civil Rights concerning electronic voting, Washington, DC, April 9, 2004.

Testimony before the U.S. House of Representatives Committee on Science concerning voting system certification, Washington, DC, June 24, 2004.

Testimony before the U.S. House of Representatives Committee on House Administration concerning voting system security, Washington, DC, July 7, 2004.

Testimony before the U.S. House of Representatives Committee on Government Reform concerning

electronic voting technology, Washington, DC, July 20, 2004.

Testimony on DREs and paper trails before the Virginia Legislature Study Commission on Voting System Certification and Security, Richmond, VA, August 16, 2004.

Testimony before the Election Assistance Commission, Technical Guidelines Development Committee, Subcommittee on Computer Security and Transparency, Gaithersburg, MD, Sept. 20, 2004.

Testimony before the House Ways and Means Committee of the Maryland General Assembly on voting machine paper trails, Annapolis, MD, December 7, 2004.

Testimony before the U.S. House of Representatives Committee on House Administration concerning paper trails, Washington, DC, September 28, 2006.

Testimony before the U.S. Election Assistance Commission concerning the Voting System Testing and Certification Program, Washington, DC, October 26, 2006.

Testimony before the Georgia State Board of Elections, Powder Springs, GA, December 21, 2007.

Testimony before the Maryland House of Delegates Ways and Means Committee, Annapolis, MD, January 18, 2007.

Testimony before the U.S. Senate Committee on Rules and Administration on the Ballot Integrity Act of 2007, Washington, DC, July 25, 2007.

Arbitration

Dr. Shamos has served as an arbitrator in computer-related disputes for the American Arbitration Association.

Electronic Voting

Dr. Shamos has served as an examiner of electronic voting systems and consultant on electronic voting.

Consultant to the Pennsylvania Secretary of the

Commonwealth (2004-).

Member, Sarasota Source Code Audit Task Force, Florida Secretary of State (2006-2007)

Consultant to the Massachusetts Secretary of the Commonwealth (2006).

Project SERVE Security Peer Review Group (2003).

Attorney General's Designee for electronic voting examinations, State of Texas (1987-2000).

Attorney for Counsel to the Secretary of the Commonwealth, Commonwealth of Pennsylvania. (1998-2000); Statutory Examiner for electronic voting, Commonwealth of Pennsylvania (1980-1996).

Consultant to Montgomery County, Pennsylvania (1996).

Consultant to the Secretary of State of Nevada (1996).

Consultant to the Delaware Legislature (1989).

Business Experience

President, Unus, Inc., database publishing software (formerly Unilogic, Ltd.) (1990-1992)

President, Lexeme Corporation (1984-87), software language translation products.

Managing Partner, Shamos and Tchen (1978-82), computer consulting firm.

Supervisory Programmer, National Cancer Institute (1970-72), while a commissioned officer in the United States Public Health Service (O-3).

Associate Engineer, IBM Corporation (1968-70), design of manufacturing information systems.

Consulting

Morgan Stanley Dean Witter (2000-2002). Contact: Stephanie Homes.

McKinsey & Co. (1999-2001). Contact: Will Draper

(BTO Stamford)

Bell Atlantic Corporation (1999-). Contact: John Martin.

LG-CNS, South Korea (2002-). Project to automate the Korean court system.

Directorships

Unilogic, Ltd. (1979–87) (later Unus, Inc. d/b/a Cygnet Publishing Technologies, 1987-). Database publishing software.

The Billiard Archive (1983–). Historical nonprofit foundation.

Lexeme Corporation (1984-1987). Computer source language translation.

Insurance Technology Corporation (1992–1995). IT consulting for the insurance industry.

Personal Data

Date of birth: April 21, 1947.

Married to Julie Shamos (formerly Julie Van Allen), August 12, 1973.

Children: Josselyn (born May 20, 1982), Alexander (born August 3, 1984).

Military Status: Veteran (Commissioned Officer, U.S. Public Health Service, 1970-72).

Health: excellent

Contact Information

Office Address:
4515 Newell Simon Hall
Carnegie Mellon University
Pittsburgh, PA 15213
Office Telephone: 412-268-8193
Office Fax: 412-268-6298
Email: shamos@cs.cmu.edu

Home Address:

605 Devonshire Street
Pittsburgh, PA 15213-2904
Home Telephone: 412-681-8398
Home Fax: 412-681-8916

Publications

SCIENCE

Books

1. *Computational Geometry: An Introduction*, with F. P. Preparata. Springer-Verlag (1985, revised ed., 1991), 390 pp. ISBN 0387961313. According to Citeseer, this is the 28th most frequently cited work in computer science.

2. *Vyichislitel'naya Geometria: Vyedyeniye*. Russian translation of "Computational Geometry: An Introduction." Moscow: Mir Publishers (1989). ISBN 5030010416.

3. *Keisan kikagaku nyumon.* Japanese translated by T. Asano and T. Asano of *Computational Geometry: An Introduction*, with F. P. Preparata. Soken Shuppan (Jul. 1992). ISBN4795263213.

4. *Handbook of Academic Titles*. 198 pp. (Nov. 2002). An encyclopedia of various academic designations used at over 1000 colleges and universities in the United States.

5. *Geometria obliczeniowa. Wprowadzenie.* Polish translation of "Computational Geometry: An Introduction." Warsaw: Helion (2003) 392 pp. ISBN 83-7361-098-7.

Book Chapters

1. "Privacy and Public Records," chapter 16 in *Personal Information Management*, Jones & Teevan, eds., Univ. of Washington Press (2007), ISBN978-0-295-98737-8.

Articles

1. "On the Piezoelectric Effect in Bone," with M. H. Shamos and L. S. Lavine. *Nature* **197**:81 (1963).

2. "An Absorber Theory of Acoustic Radiation," with M. A. Tavel. *Journal of the Acoustical Society of America* **54**:46–49 (1973).

3. "Problems in Computational Geometry." Unpublished book manuscript (1974, revised 1977). Distributed in photocopy.

4. "Geometric Complexity." *Proceedings of the Seventh Annual*

ACM Symposium on Automata and Theory of Computation (May 1975) 224–233.

5. "Closest-point Problems," with D. J. Hoey. *Proceedings of the Sixteenth IEEE Symposium on Foundations of Computer Science* (Oct. 1975) 151–162.

6. "Divide and Conquer in Multidimensional Space," with J. L. Bentley. *Proceedings of the Eighth Annual ACM Symposium on Automata and Theory of Computing* (May 1976) 220–230.

7. "Geometric Intersection Problems," with D. J. Hoey. *Proceedings of the Seventeenth Annual IEEE Symposium on Foundations of Computer Science* (Oct. 1976) 208–215.

8. "Lower Bounds from Complex Function Theory," with G. Yuval. *Proceedings of the Seventeenth Annual IEEE Symposium on Foundations of Computer Science* (Oct. 1976) 268–273.

9. "Geometry and Statistics: Problems at the Interface." In *Algorithms and Complexity: New Directions and Recent Results*, J. F. Traub, ed., Academic Press (1976) 251–280.

10. "Divide and Conquer for Linear Expected Time," with J. L. Bentley. *Information Processing Letters* 7 (1977) 87–91.

11. "A Problem in Multivariate Statistics: Algorithm, Data Structure, and Applications," with J. L. Bentley. *Proceedings of the Fifteenth Allerton Conference on Communications, Control and Computers* (Sep. 1977) 193–201.

12. "Optimal Algorithms for Structuring Geographic Data," with J. L. Bentley. *Proceedings of the Harvard Conference on Topological Data Structures for Geographic Information Systems* (Oct. 1977) 43–51.

13. "Computational Geometry." Ph.D. Thesis, Yale University (1978). University Microfilms, Ann Arbor, MI.

14. "On Time and Space," with A. R. Meyer. In *Perspectives on Computer Science*, A. K. Jones, ed. Academic Press (1978).

15. *Combinatorics on Graphs I: Graph Polynomials*. Unpublished book manuscript (1978).

16. "Robust Picture Processing Operators and Their Implementation as Circuits." *Proceedings of the Fall 1978 Workshop on Image Processing*, Carnegie Mellon University (1978).

17. "A practical system for source language translation," with T. R. Kueny and P. L. Lehman. *Proceedings of the National Conf. on Software Reuseability and Maintainability*, pp. B-1 – B-12, Washington, DC (Sep. 1986).

18. "The Early Years of Computational Geometry – A Personal Memoir." *Advances in Discrete and Computational Geometry* (B. Chazelle, J. E. Goodman, and R. Pollack, eds.), *Contemporary Mathematics*, Amer. Math. Soc., Providence (1998).

DIGITAL LIBRARIES

Articles

1. "Machines as readers: a solution to the copyright problem." J. Zhejiang Univ. Science 6A, 11, pp. 1179-1187 (Nov. 2005).

Reports

1. "Japanese Digital Information Policy, Intellectual Property and Economics," in "Digital Information Organization in Japan," International Technology Research Institute (1998).

ELECTRONIC VOTING

Articles

1. "Voting System Certification — An Examiner's View." Invited paper presented at the Election Center Conference, Reno, Nevada (Sep. 1989).

2. "Electronic Voting — Evaluating the Threat." Proc. Third ACM Conf. on Computers, Freedom & Privacy, San Francisco, CA (Mar. 1993).

3. "Paper v. Electronic Voting Records — An Assessment." Proc. 14th ACM Conf. on Computers, Freedom & Privacy, Berkeley, CA (Apr. 2004).

4. "Evaluation of Voting Systems," with P.L. Vora, B. Adida, R. Bucholz, D. Chaum, D. Dill, D. Jefferson, D. Jones, W. Lattin, A. Rubin and M. Young, Commun. ACM 47(11):144 (2004).

5. "Voting as an Engineering Problem." *The Bridge* (publication of the National Academy of Engineering), Summer 2007, pp. 35-39.

BILLIARDS

Books

1. *Pool.* New York: Mallard Press division of Bantam-Doubleday-Dell Promotional Book Company (Aug. 1991). 128 pp. ISBN 0-7924-5310-7.

2. *Le billard et le billard américain*. Paris: Minerva, 1992, reprinted 1997. 128 pp. Translation by Jean-Yves Prate of the author's American book, *Pool*. ISBN 2–8307–0160–7 (1992), 2-8814-3135-6 (1997).

3. *The Illustrated Encyclopedia of Billiards*. New York: Lyons & Burford (1993). 310 pp. ISBN 1-55821-219-1.

4. *Pool Snooker Carambola*. Padua: Facto Edizioni (1993). 128 pp. Italian translation of *Pool*. Translated by Elisabetta Bezzon. ISBN 88-85860-20-6. The only English-language billiard book ever published in Italian.

5. *Pool*. New York: Friedman/Fairfax (Jun. 1994). 128 pp. ISBN 1-56799-061-4. Paperback edition of the author's 1991 *Pool*.

6. *Shooting Pool: The People, the Passion, the Pulse of the Game,* with photographs by George Bennett. New York: Artisan (Jun. 1998). 144 pp. ISBN 1-885183-95-X. A Book-of-the-Month Club bonus selection (Fall, 1998).

7. *Setting the Stage for Fifty Years*. Coralville, IA: Billiard Congress of America (Jun. 1998). 88 pp. A history of the Billiard Congress of America.

8. *The New Illustrated Encyclopedia of Billiards*. New York: Lyons Press (1999). 320 pp. ISBN 1-55821-797-5. An expanded and revised edition of *The Illustrated Encyclopedia of Billiards*.

9. *The Complete Book of Billiards*. New York: Gramercy Books (2000). 306 pp. ISBN 0-517-20869-5. Reissue of author's 1993 *The Illustrated Encyclopedia of Billiards*.

In Preparation

SCIENCE

Books

1. *A Catalog of the Real Numbers*. A list, patterned after Sloane & Plouffe, The Encyclopedia of Integer Sequences, Academic Press (1995). Over 8000 interesting real numbers arranging in lexical order by decimal expansion, with accompanying formulas.

2. *Handbook of Academic Titles*.

Articles

1. *Overcounting Functions.* A systematic method of transforming certain multiple summations into single summations, with new number-theoretic results.

2. *Property Enumerators and a Partial Sum Theorem*. A new result allowing rapid symbolic evaluation of certain types of double summations.

LAW

Books

1. *A Dictionary of American Intellectual Property*.

2. *Electronic Voting Glossary*. To be published by NIST.

Invited Talks

ELECTRONIC COMMERCE

"The U.S., Korea and the Internet Bubble." Korea International Trade Association (Seoul, July 2003).

"Electronic Judiciary Services in the United States." Address at the Supreme Court of Korea (Dec. 2004).

"eGovernment in the United States." Public address at the University of Hong Kong (Feb. 2005).

"Global SCM as a Cross-Border eCommerce Model," Korea International Trade Association, Seoul, Korea (Mar. 2007).

SCIENCE AND LAW

"Digital Property in the 21st Century." Keynote address for the Spring Meeting of the American Intellectual Property Law Association, Pittsburgh, PA (May 2000). View slides.

"Who Owns This Algorithm?" Carnegie Mellon University (Nov 1991); Microelectronics and Computer Corporation (Jan. 1992); Univ, of Texas at Austin (Jan. 1992); UCLA (Feb. 1992).

"New Computer Technology and Its Application to Worker's Compensation." Forum IV, Newport Beach, CA (Feb. 1992).

"The Office of the Future, If There Is One." 1994 IAIABC Conf., Pittsburgh, PA (Sep. 1994).

"The Fringes of Infringement." University of Texas, Austin, TX (Sep. 1995).

"The Arts and the Internet." Allegheny County Bar Association Continuing Legal Education course (June 26, 1996).

"The Universal Information Resource." Inventing the Future, Symposium in Honor of Raj Reddy's 60th Birthday, Carnegie Mellon University, Pittsburgh, PA (May 1998).

"The Universal Library." University of Texas at Austin (Sep. 1998)

"The Universal Library and Its Role in Scientific Information." Keynote address to the RNA Society symposium on Emerging Sources of RNA Information, Arlington, VA (Dec. 8, 1998).

"Digital Property in the 21st Century." Luncheon address to the American Intellectual Property Law Association, Pittsburgh, PA (May. 2000).

"The Future of eCommerce." Address to the Association for Corporate Growth, Pittsburgh, PA (Dec. 2001).

"Copyright Protection and Distance Learning." Hong Kong Intellectual Property Office (Feb. 2002).

"Surprises in Experimental Mathematics." Carnegie Mellon University Mathematics Seminar (Feb. 2002).

"The Universal Dictionary." Address at International Institute of Information Technologies (IIIT), Hyderabad, India (Jan. 2003).

"The Million Book Projects." Public address at the University of Hong Kong (Jan. 2003).

"Mathematics and the Privacy Laws." ALADDIN Workshop on Privacy in D.A.T.A., Pittsburgh, PA (Mar. 2003).

"Machines as readers: a solution to the copyright problem." 1st Int'l Conf. on Universal Digital Library, Hangzhou, China (Nov. 2005).

"University Technology Transfer: How to Fix It." Asia Conference on Technology Transfer (ACTT) 2006, Seoul, S. Korea (Mar. 2006).

"How Big a Problem is Copyright"? USAIN Conference, Cornell University, Ithaca, NY (Oct. 2006).

"Digital Ownership." 2d Intl. Conf. on Universal Digital Library, Alexandria, Egypt (Nov. 2006).

ELECTRONIC VOTING

"Voting System Certification — An Examiner's View." Election Center Conference, Reno, Nevada (Sep. 1989).

"Electronic Voting — Evaluating the Threat." Third Conf. on Computers, Freedom and Privacy, San Francisco, CA (Mar. 1993).

"What's Happing in Florida?" Carnegie Mellon University (Nov. 2001)."

"Electronic Voting: The Technology of Democracy." Hong Kong University (Feb. 2004).

"Theory v. Practice in Electronic Voting." DIMACS (Rutgers Univ., May 2004).

"HAVA: Are We Ready?" Panel at the League of Women Voters National Convention, Washington, DC (Jun. 2004).

"Testing Voting Machines." Panel at the American Enterprise Institute, Washington, DC (Jun. 2004).

"Electronic Voting: Promise and Peril." Talk at the Moritz College of Law, Ohio State University (Sep. 2004).

"Is e-voting ready for prime time: Legal and technical issues regarding the upcoming Presidential election." Panel at John Marshall Law School (Chicago, IL, Oct. 2004).

"Is Electronic Voting Reliable?" Talk to the Kiwanis Club of Dubuque, Iowa (Feb. 2005).

"The Top Ten Problems in Practical Electronic Voting." Int'l Workshop on Mathematics and Democracy, Ettore Majorana Centre, Erice, Sicily (Sept. 2005).

"Why Don't We Have Paper Trails in Pennsylvania?" Carnegie Mellon Univ. CyLab Seminar, Pittsburgh, PA (Jan 2006).

"Paper Trails and the Pennsylvania Certification Process." County Commissioners Association of Pennsylvania 2006 Spring Conference, Harrisburg, PA (Mar. 2006).

"The 2006 Elections: Are We Ready?" Panel at the American Enterprise Institute, Washington, DC (Sept. 2006).

"What's Right with Electronic Voting?" University Lecture Series, Carnegie Mellon University (Oct. 12, 2006).

"What Happened in Yesterday's Election?" Center for Research on Computation and Society, Harvard University (Nov. 8, 2006).

"What Happened in Sarasota County"? Council on Government Ethics Laws, New Orleans, LA (Dec. 6, 2006).

"What Happened to 18,000 Votes? Results of the Sarasota Source Code Audit." Carnegie Mellon University (Apr. 16, 2007).

A32

IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION

EPICREALM LICENSING, LP, Plaintiff, v. AUTOFLEX LEASING, INC., et al., Defendants.	§ § § § § § § § §	2:05-CV-163

EPICREALM LICENSING, LP, Plaintiff, v. FRANKLIN COVEY CO., et al., Defendants.	§ § § § § § § § §	2:05-CV-356

JOINT CLAIM CONSTRUCTION AND PRE-HEARING STATEMENT

Pursuant to Patent Rule 4-3 and the Docket Control Order, the parties hereby submit this Joint Claim Construction and Pre-hearing Statement. As reflected in the Certificate of Conference below, all parties have agreed to this Joint Claim Construction and Pre-hearing Statement, and plaintiff epicRealm Licensing, LP ("epicRealm") is filing it on behalf of itself and all defendants.

I. Agreed Constructions

The parties have agreed upon the construction of the following claim terms, phrases, and clauses appearing in U.S. Patent No. 5,894,554 ("the '554 patent") and U.S. Patent No. 6,415,335 ("the '335 patent").

Agreed Claim Constructions

Item	'554 Claims	'335 Claims	Agreed Construction
appropriate page server		29	a page server determined to be capable of processing said request based on said dynamic information
connection cache	4	5, 19	a store of information identifying active connections
data dynamically retrieved	1, 9	1, 10, 15, 24, 29	data retrieved in response to a request
data sources, one or more	1-5, 8-9, 11	1, 3-6, 9-10, 12, 15, 17-20, 23-24, 26, 29	databases, spreadsheets, files and any other type of data repository
drives a format of the data		10, 24	determines a format of the data
dynamic Web page	1, 9, 11	1	a Web page that is created in response to a request
dynamically generating	1, 3, 6, 9, 11	1, 4, 7, 15, 18, 21, 29	creating in response to a request
dynamically retrieving	3	4, 18	retrieving in response to a request
dynamically updating data		12, 26	writing data in response to a request
OLE extension		14, 28	Microsoft Object Linking and Embedding extension; a technology defined by Microsoft for transferring and sharing information among applications
other requests	1, 9, 11	1, 15, 29	different requests
requests	1, 9, 11	1, 15, 29	plural form of the term "request," as construed by the court

II. Disputed Constructions

At this time, the parties do not agree upon the construction of the following claim terms, phrases, and clauses, which were identified by one or more parties as requiring construction.

Item No.	Item	Plaintiff's Construction	Intrinsic Support	Extrinsic Support
21	HTML templates	*epicRealm submits that this term or phrase needs no construction because it has a plain and ordinary meaning. Should the court decide a construction is needed, epicRealm's proposed construction is:* "an HTML model that specifies the use of markers as place holders for content"	6:33-49	
22	HTTP-compliant device	a device that understands HTTP and whose behavior is affected by an HTTP request	1:23-26; FIG. 2; FIG. 4; 2:64-5:36; 6:20-32; 8:63-9:11	• Oxford English Dictionary - "HTTP", "compliant" at http://dictionary.oed.com/ • The American Heritage® Dictionary of the English Language - "HTTP", "compliant" at http://www.bartleby.com/61/
27	intercepting	stopping, deflecting, or interrupting the processing of a request	FIG. 5 (step 504); 8:26-32; FIG. 4; 4:54-5:7	• Oxford English Dictionary - "Intercept" at http://dictionary.oed.com/ • Merriam-Webster Online - "Intercept" at http://www.m-w.com/ • The American Heritage® Dictionary of the English Language - "Intercept" at http://www.bartleby.com/61/ • The American Heritage Dictionary (1996)
29	interceptor	*epicRealm submits that this term or phrase needs no construction because the term "intercepting" is being construed. Should the Court decide a construction is needed, epicRealm's proposed construction is:* "a system that performs intercepting"	FIG. 5 (step 504); 8:26-32; FIG. 4; 4:54-5:7	• Oxford English Dictionary - "Intercept" at http://dictionary.oed.com/• Merriam-Webster Online - "Intercept" at http://www.m-w.com/• The American Heritage® Dictionary of the English Language - "Intercept" at http://www.bartleby.com/61/ • The American Heritage Dictionary (1996)

Item No.	Item	Plaintiff's Construction	Intrinsic Support	Extrinsic Support
42	processing said request, said processing being performed by said page server while said Web server/HTTP-compliant device concurrently processes said other requests	*epicRealm submits that this term or phrase needs no construction because the following terms are already being construed: "request", "page server", "Web server", "HTTP-compliant device." The Parties have agreed to the construction of "other requests." The remaining terms or phrases, including "concurrently processes," have plain and ordinary meaning. Should the Court decide a construction is needed for "concurrently processes", epicRealm's proposed construction is:* "processes at the same time"	Abstract; 2:20-35; 4:11-53; 5:8-19; 6:20-32 9:6-8; 10:37-39	• Oxford English Dictionary - "Request", "Server", "Web Server", "HTTP", "Process", "Perform", "Concurrent" at http://dictionary.oed.com/ • Merriam-Webster Online - "Request", "Server", "Process", "Perform", "Concurrent" at http://www.m-w.com/ • The American Heritage® Dictionary of the English Language - "Request", "Server", "HTTP", "Process", "Perform", "Concurrent" at http://www.bartleby.com/61/ • RedHat Glossary - "Request" at www.redhat.com/docs/manuals/stronghold/Stronghold-3.0-Manual/admin-guide/glossary.fm.html • WebObjects Glossary - "Request" at developer.apple.com/documentation/WebObjects/WebObjects_Overview/Glossary/chapter_10_section_1.html • Newton's Telecom Dictionary (11th edition) (1996) - "HTTP"
43	releasing	freeing a process to service other requests	4:38-53; 5:8-20; 6:21-27	• Oxford English Dictionary - "Release" at http://dictionary.oed.com/ • Merriam-Webster Online - "Release" at http://www.m-w.com/ • The American Heritage® Dictionary of the English Language - "Release" at http://www.bartleby.com/61/

A33

IN THE UNITED STATES DISTRICT COURT
OF THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION

EPICREALM, LICENSING, LLC	§	
	§	
v	§	**No. 2:05CV163**
	§	
AUTOFLEX LEASING, INC., et al.	§	

EPICREALM, LICENSING, LLC	§	
	§	
v	§	**No. 2:05CV356**
	§	
FRANKLIN COVEY CO., et al.	§	

ORDER ADOPTING REPORT AND RECOMMENDATION REGARDING CLAIM CONSTRUCTION

The above-entitled and numbered civil action was heretofore referred to United States Magistrate Judge Caroline M. Craven pursuant to 28 U.S.C. § 636. The Report and Recommendation Regarding Claim Construction of the Magistrate Judge which contains her proposed findings of fact and recommendations for the disposition of such action has been presented for consideration. Plaintiff and Defendants both filed objections to the Report and Recommendation of the Magistrate Judge. The Court conducted a *de novo* review.

The Court, having reviewed the relevant briefing, finds the parties' objections are without merit. In their briefing, the parties address, among other things, the word "mechanism" as used in the Magistrate Judge's discussion of the construction of "Web page." *See* Report and Recommendation at pgs. 8-9 ("As described within the

specification, a Web page is a mechanism through which static and dynamic content may be displayed.")(emphasis added). A Web page may include static or dynamic content. However, the Magistrate Judge's construction of "Web page" as "Web content displayable through a Web browser" does not include the word "mechanism." The Court adopts the construction of "web page" as proposed by the Magistrate Judge, and with the exception of the use of the word "mechanism," the Court also adopts the reasoning of the Magistrate Judge with respect to the construction of "Web page."

The Court is of the opinion that the findings and conclusions of the Magistrate Judge are correct. Therefore, the Court hereby adopts the Report of the United States Magistrate Judge as the findings and conclusions of this Court.

IT IS SO ORDERED.

SIGNED this 30th day of October, 2006.



DAVID FOLSOM
UNITED STATES DISTRICT JUDGE

IN THE UNITED STATES DISTRICT COURT
OF THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION

EPICREALM, LICENSING, LLC	**§**	
	§	
v	**§**	**No. 2:05CV163**
	§	
AUTOFLEX LEASING, INC., et al.	**§**	

EPICREALM, LICENSING, LLC	**§**	
	§	
v	**§**	**No. 2:05CV356**
	§	
FRANKLIN COVEY CO., et al.	**§**	

REPORT AND RECOMMENDATION
REGARDING CLAIM CONSTRUCTION

Pursuant to the provisions of 28 U.S.C. § 636(b)(1) and (3) and the Amended Order for the Adoption of Local Rules for Assignment of Duties to United States Magistrate Judges, the above-entitled and numbered cause of action was referred to the undersigned for pretrial purposes. Claim construction arguments in cause numbers 2:05-CV-163 and 2:05-CV-356 were combined, and Defendants submitted joint briefing. The Court conducted a claim construction hearing on July 13, 2006. This Report and Recommendation construes certain terms in United States Patent Nos. 5,894,554 ("the '554 Patent) and 6,415,335 (the '335 Patent).

I. BACKGROUND

The '554 Patent issued on April 13, 1999. The '335 Patent issued on July 2, 2002 and is a divisional application of the '554 Patent. The '554 Patent and the '335 Patent share a common specification.[1] The patents generally relate to managing Web sites. More particularly, the patents relate to managing dynamic Web page generation. Col. 2:15-23. The patents distinguish some Web pages as having a static nature that remains static until manually modified and other Web pages as being dynamic Web pages which contain content that is generated dynamically by retrieving the necessary requested data and generating the requested Web page dynamically. Col. 1:38-55. The patents describe prior art Web servers as handling both static and dynamic Web page requests. Col. 3:64 - Col. 4:37; Figures 2-3. The techniques described in the patents include routing a Web request from the Web server to a Page server. The Page server may than process the request, and the Web server is released to process other requests. Col. 2:20-35; Col. 4:54 - Col. 6:32. In this manner, dynamic Web pages may be generated by the Page servers.

Some of the claim construction disagreements involve common themes. For example, in general the Plaintiff construes various terms so that Web servers and Page servers do not have to be separate machines while the Defendants seek constructions that would include a separate machine concept. The Plaintiff also seeks constructions that do not include the concept of Uniform Resource Locators (URLs) while the Defendants add the term URL to some of the constructions they seek. Other conflicting claim construction positions are more specific to individual terms that are in dispute.

[1] References to the specification will refer to the column and line numbers of the '554 Patent.

II. APPLICABLE LAW

"It is a 'bedrock principle' of patent law that 'the claims of a patent define the invention to which the patentee is entitled the right to exclude.'" *Phillips v. AWH Corp.*, 415 F.3d 1303, 1312 (Fed. Cir. 2005) (en banc) (*quoting Innova/Pure Water Inc. v. Safari Water Filtration Sys., Inc.*, 381 F.3d 1111, 1115 (Fed. Cir. 2004)). In claim construction, courts examine the patent's intrinsic evidence to define the patented invention's scope. *See id.*; *C.R. Bard, Inc. v. U.S. Surgical Corp.*, 388 F.3d 858, 861 (Fed. Cir. 2004); *Bell Atl. Network Servs., Inc. v. Covad Commc'ns Group, Inc.*, 262 F.3d 1258, 1267 (Fed. Cir. 2001). This intrinsic evidence includes the claims themselves, the specification, and the prosecution history. *See Phillips*, 415 F.3d at 1314; *C.R. Bard, Inc.*, 388 F.3d at 861. Courts give claim terms their ordinary and accustomed meaning as understood by one of ordinary skill in the art at the time of the invention in the context of the entire patent. *Phillips*, 415 F.3d at 1312-13; *Alloc, Inc. v. Int'l Trade Comm'n*, 342 F.3d 1361, 1368 (Fed. Cir. 2003).

The claims themselves provide substantial guidance in determining the meaning of particular claim terms. *Phillips*, 415 F.3d at 1314. First, a term's context in the asserted claim can be very instructive. *Id*. Other asserted or unasserted claims can also aid in determining the claim's meaning because claim terms are typically used consistently throughout the patent. *Id*. Differences among the claim terms can also assist in understanding a term's meaning. *Id*. For example, when a dependent claim adds a limitation to an independent claim, it is presumed that the independent claim does not include the limitation. *Id*. at 1314–15.

Claims "must be read in view of the specification, of which they are a part." *Id*. (quoting *Markman v. Westview Instruments, Inc.*, 52 F.3d 967, 978 (Fed. Cir. 1995)). "[T]he specification 'is always highly relevant to the claim construction analysis. Usually, it is dispositive; it is the single

best guide to the meaning of a disputed term.'" *Id.* (*quoting Vitronics Corp. v. Conceptronic, Inc.*, 90 F.3d 1576, 1582 (Fed. Cir. 1996)); *Teleflex, Inc. v. Ficosa N. Am. Corp.*, 299 F.3d 1313, 1325 (Fed. Cir. 2002). This is true because a patentee may define his own terms, give a claim term a different meaning than the term would otherwise possess, or disclaim or disavow the claim scope. *Phillips*, 415 F.3d at 1316. In these situations, the inventor's lexicography governs. *Id.* Also, the specification may resolve ambiguous claim terms "where the ordinary and accustomed meaning of the words used in the claims lack sufficient clarity to permit the scope of the claim to be ascertained from the words alone." *Teleflex, Inc.*, 299 F.3d at 1325. But, "although the specification may aid the court in interpreting the meaning of disputed claim language, particular embodiments and examples appearing in the specification will not generally be read into the claims." *Comark Commc'ns, Inc. v. Harris Corp.*, 156 F.3d 1182, 1187 (Fed. Cir. 1998); *see also Phillips*, 415 F.3d at 1323. The prosecution history is another tool to supply the proper context for claim construction because a patent applicant may also define a term in prosecuting the patent. *Home Diagnostics, Inc., v. Lifescan, Inc.*, 381 F.3d 1352, 1356 (Fed. Cir. 2004) ("As in the case of the specification, a patent applicant may define a term in prosecuting a patent.").

Although extrinsic evidence can be useful, it is "less significant than the intrinsic record in determining 'the legally operative meaning of claim language.'" *Phillips*, 415 F.3d at 1317 (*quoting C.R. Bard, Inc.*, 388 F.3d at 862). Technical dictionaries and treatises may help a court understand the underlying technology and the manner in which one skilled in the art might use claim terms, but technical dictionaries and treatises may provide definitions that are too broad or may not be indicative of how the term is used in the patent. *Id.* at 1318. Similarly, expert testimony may aid a court in understanding the underlying technology and determining the particular meaning of a term

in the pertinent field, but an expert's conclusory, unsupported assertions as to a term's definition is entirely unhelpful to a court. *Id*. Generally, extrinsic evidence is "less reliable than the patent and its prosecution history in determining how to read claim terms." *Id*.

The patents in suit also contain means-plus-function limitations that require construction. Where a claim limitation is expressed in "means plus function" language and does not recite definite structure in support of its function, the limitation is subject to 35 U.S.C. § 112, ¶ 6. *Braun Med., Inc. v. Abbott Labs.*, 124 F.3d 1419, 1424 (Fed. Cir. 1997). In relevant part, 35 U.S.C. § 112, ¶ 6 mandates that "such a claim limitation 'be construed to cover the corresponding structure . . . described in the specification and equivalents thereof.'" *Id*. (citing 35 U.S.C. § 112, ¶ 6). Accordingly, when faced with means-plus-function limitations, courts "must turn to the written description of the patent to find the structure that corresponds to the means recited in the [limitations]." *Id*.

Construing a means-plus-function limitation involves multiple inquiries. "The first step in construing [a means-plus-function] limitation is a determination of the function of the means-plus-function limitation." *Medtronic, Inc. v. Advanced Cardiovascular Sys., Inc.*, 248 F.3d 1303, 1311 (Fed. Cir. 2001). Once a court has determined the limitation's function, "the next step is to determine the corresponding structure disclosed in the specification and equivalents thereof." *Id*. A "structure disclosed in the specification is 'corresponding' structure only if the specification or prosecution history clearly links or associates that structure to the function recited in the claim." *Id*. Moreover, the focus of the "corresponding structure" inquiry is not merely whether a structure is capable of performing the recited function, but rather whether the corresponding structure is "clearly linked or associated with the [recited] function." *Id*.

III. DISCUSSION

A. Disputed Claim Terms

1. "Web Page"

"Web page" is utilized in asserted claims 1, 3, 6, 7, 9, and 11 of the '554 Patent and 1, 4, 7, and 8 of the '335 Patent. The Plaintiff asserts that "Web page" does not need construction. Alternatively, if construed, the Plaintiff asserts that the proper construction of the term is "content displayable through a Web browser." The Defendants assert that the terms should be construed as "an HTML document accessible through a URL."

The Court first notes that the Plaintiff's original briefing expressed concern that the Defendants construction implies that dynamically generated Web pages are not included within the term "Web page." The Court finds this concern somewhat unfounded, and it is noted that the Defendants clearly referred to "Web page" in their briefing and oral argument as encompassing both static and dynamic Web pages.

The other assertions by the parties primarily revolve around two issues, the inclusion of terms "HTML" and "URL" in the construction. The Plaintiff argues that the term "content" is utilized in the specification at least twice to describe what is displayed on a Web page. Col. 1:47-51; Col. 7:23-26. Further, the Plaintiff points out that in the Defendants' own briefing Web pages are referred to as containing content. Defendants' Brief at 11-12. The Plaintiff further asserts that the Defendants are also attempting to read in limitations from the specification and that such a construction would exclude documents formatted in other formats such as SGML, XHTML, XML, and JPG.

The Defendants state that HTML is a software language and argue that as described within the specification HTML documents are what are sent back as Web pages. Col. 1:18-22; Figures 3

and 5. The Defendants also cite a Microsoft Press Computer Dictionary definition which states that "A Web page consists of an HTML file…" Defendants' Brief at 9. The Defendants further assert that "content" in the specification refers to information included in a Web page and that such content itself does not form a Web page.

The Court notes that the specification does appear to consistently refer to the HTML language and does not mention other software languages. However, the Defendant does not identify persuasive support within the specification that the invention must be limited to only one type of software language. Moreover, Defendants have not persuaded the Court that in light of the specification one skilled in the art would assume a Web page as referred to in the patents could only be generated with the HTML language. Further, upon review of the whole specification and claims, with respect to Web pages the described concepts are not related to the intricacies of what particular programming languages are used to display a Web page but rather merely the higher level differentiation of static pre-existing Web pages verse dynamically generated Web pages. In these circumstances, the Court finds it improper to incorporate the limitation of HTML within the more general term Web page that is utilized in the claims themselves.

With regard to the URL concept, the Plaintiff asserts that the Defendants' definition adds additional complexity to the claim construction as the meaning and scope of "accessible through a URL" could itself require construction. Further, citing the extrinsic evidence that the Defendants themselves put before the Court, the Plaintiff argues that it is known that when a Web browser sends a request what is actually sent does not match what is commonly known as a full URL. In oral argument the Plaintiff asserted that the Defendants' extrinsic evidence shows a URL as "http_URL ="http:" "//" host [":" port] [abs_path] thus requiring four components: a protocol (HTTP), a host,

a port and an absolute path. Further the Plaintiffs argue that this same extrinsic evidence shows that a request most commonly is structured for example "GET / pub / WWW / TheProject.html HTTP / 1.0" The Plaintiff asserts that this also emphasizes the concern over the ambiguity of the meaning of "accessible through a URL."

The Defendants turn to the specification, which includes the statement "[a] URL is a Web address that identifies the Web page and its location on the Web." Col. 1:30-33. The Defendants also note the language that states "[w]hen the appropriate Web site receives the URL, the Web page corresponding to the requested URL is located…." Col. 1:33-34. Further, the Defendants point to other examples in the specification, such as Figure 3, which refer to the Web browser sending the URL request. At oral argument, the Defendants also stated that what is sent by a Web browser does include the URL information.

The arguments of the parties highlight some of the concerns the Court has with the inclusion of the term URL. A definition of the meaning of URL within the usage of the patent would further be necessitated as URL is alternatively referred to as "what is examined by the Web browser," a URL request is received by the Web server, and a URL request can be sent to a page server. Col. 4:13-14; Col. 8:28-32; Col. 8:38-39. The specification does not make clear what is the particular structure and content that is meant by the use of the term "URL" at each of these stages of the process. Again, as with the HTML term, this is not surprising as the specification and claims as a whole do not focus on the particular type of request that is made or the particular structure/content of a request as it processed through the system beyond the static and dynamic distinction discussed above. Further, to add the term "accessible" to the construction would necessitate further claim construction as to what "accessible" means. As described within the specification, a Web page is

a mechanism through which static and dynamic content may be displayed. The particular addressing mechanism at each step of the processing of a dynamic Web page is not noted in the specification to be a requirement or of particular importance to the claimed invention. An inclusion of the term URL would improperly incorporate limitations from the specification for the term Web page which has a meaning that is adequately described within the full context of the specification.[2] **Thus, the Court construes "Web page" to mean "Web content displayable through a Web browser."**

2. "Request"

"Request" is utilized in asserted claims 1, 9, 11 of the '554 Patent and 1, 15, and 29 of the '335 Patent. The Plaintiff asserts that the proper construction of "request" is "a message that asks for content." The Defendants assert that the term should be construed as "a message containing a URL that asks for a Web page specified by the URL."

The Court first notes that the term "request" generally appears in the claims in different circumstances. In some claims (for example claim 1 of the '554 Patent), request is first utilized with relation to "a dynamic Web page generation request" and multiple references are then made to "said request." Such claims also use the term "other requests" which imply a request different from "said request." Other claims (for example claim 15 of the '335 Patent) begin with the general use of "a request" but later refer to "dynamically generating a page in response to said request." These claims also refer to "other requests." As noted in the specification, a request can refer to static Web

[2] Additional discussion regarding the inclusion of URL limitations is also provided below with regard to the term "request."

pages (such as a static document) or dynamic Web pages (such as a Web page dynamically generated by an application). Col. 1:38-56, Col. 4:16-32 Thus, in its general use, request is not limited to dynamic or static requests. It is also noted that both types of claims consistently refer to either "Web page" generation or "page" generation. The specification uniformly refers to pages in the context of Web pages. Thus, it is not unreasonable that in light of the consistent specification one would interpret "requests" as relating to requests for Web pages.

The primary point of distinction between the proposed constructions relates to inclusion of "URL" in the construction of request. The specification notes that "a Web browser allows a Web client to request a particular Web page from a Web site by specifying a Uniform Resource Locator (URL). A URL is a Web address that identifies the Web page and its location on the Web." Col. 1:29-32

The Defendants cites numerous places in the specification text in which "URL request" is utilized to refer to what is requested. The Plaintiff asserts that a request may be made at multiple points, such as shown in Figure 4 between the Web client and Web server, between the Web server and the Dispatcher and between the Dispatcher and the Page servers. The Plaintiff further argues that once a message is received by a Web server a full URL is not utilized subsequently, such as by Page servers.

The Defendants counter by noting numerous specification citations to the term "URL request" including "route the URL request to a Page server" and "the dispatcher sends the URL request to an appropriate Page server." Col. 6:9-10, Col. 8:38-39, *See* Defendants' Brief at 13-14. The Plaintiff responds that there is no teaching in the specification that a page server utilizes a URL.

Moreover, the Plaintiffs assert that it would be illogical for a request provided to the page server to utilize a URL as such address would refer to the Web server.

The Court notes that within the claims "request" is used in context of multiple steps of the page generation process. For example, within claim 1 a request may be provided to a Web server while "said request" is also received by a Page server. The specification provides varied and not always consistent uses of the terms "request" and "URL request." As noted above, URL request is often utilized. However, the more general term "request" is also often utilized. Col. 2:1-12; Col. 2:18-35; Col. 4:33-53; Col. 5:8-59; Col. 6:20-32; Col. 7:5-6. In at least two of these instances, language stating "requests or 'hits'" is utilized. Col. 4:38-39; Col. 7:5-6. Further, the Defendants have not shown within the specification that a Page server utilizes a URL. Thus, even when "URL request" is provided in the specification it is not clear that such request is required to contain a URL or is merely a request generated from an initial URL provided at a Web client. To require "request" to include a URL would thus include limitations that the specification does not clearly support and clearly require.

The remaining dispute between the parties relates to the use of "content" verse "Web page." This dispute relates to the underlying meaning of the term Web page as discussed above and thus does not need to be re-addressed. As the Court has noted, the context of the patent is uniformly directed towards Web page requests. Under the guidance provided in *Phillips*, it is appropriate when viewing the specification and the language of the claims themselves to limit "request" to Web page applications. **Thus, the Court construes "request" to mean "a message that asks for a Web page" (with the term Web page having the construction provided herein)**

3. "Page Server"

"Page server" is utilized in asserted claims 1, 4, 7, and 9-11 of the '554 Patent and 1, 2, 5, 8, 15-16, 19, 22 and 29 of the '335 Patent. The Plaintiff asserts that the proper construction of "page server" is "a processing system operable to receive a request and dynamically generate content in response to the request." The Defendants assert that the term should be construed as "page-generating software that generates a dynamic Web page on a machine separate from the Web server machine." In the claim construction Oral Argument, the Plaintiff agreed to the use of "page-generating software" in place of the term "a processing system" as previous proposed by the Plaintiff. The differences between the parties with regard to the use of "content" verse "Web page" are rooted in the basic dispute over the meaning of Web page as discussed above. Both parties include the concept of dynamic generation in their proposed constructions.

In the post oral hearing briefing, the parties each acknowledged that the primary dispute regarding the construction of "page server" is whether the Page server has to be on a machine separate from the Web server. As discussed below, the Court agrees with the Plaintiff with regard to this point of dispute.

Each party points to the specification to support their asserted position. The Plaintiff asserts that the specification includes statements that indicate that the Page server could operate on the same machine as the Web server. In particular, the Plaintiffs have pointed to passages which state:

> FIG. 1 illustrates a typical computer system 100 in which the present invention operates. Col. 2:66-67.
>
> The preferred embodiment of the present invention is implemented as a software module, which may be executed on a computer system such as computer system 100 in a conventional manner. Col. 3:55-58.

Figure 1 illustrates a computer system having a processor, bus, memory and mass storage. Further, it is stated that "the preferred embodiment of the present invention" may be implemented on a personal computer or alternatively a workstation. Col. 2:67-Col. 3:5. The Plaintiff asserts that this language is consistent with the specification as a whole by asserting that the specification describes a partitioned software architecture in which in some embodiments the software modules may all reside on the same machine and in other embodiments the software modules may reside on different computers.

The Plaintiff also points to a passage that describes an embodiment that does not have the advantage of "off-loading the processing of Web requests from the Web server machine" to a separate machine. Col. 5:26-36. However, the Court notes that this passage makes specific reference to the division between a Web server and a Dispatcher, and it is not clear in this passage alone that the Page server is also included in this use of a single machine.

The Defendants argue that the specification describes a distinction over the prior art that amounts to an explicit characterization of the invention that disclaims the prior art. *See SciMed Life Sys. V. Advanced Cardiovascular Sys.*, 242 F.2d 1337, 1343 (Fed. Cir. 2001). In particular, the Defendants point to a passage of the specification that describes the multi-threading techniques of prior art Web servers. Col. 4:32-53. This passage concludes with "[t]he claimed invention addresses this need by utilizing a partitioned architecture to facilitate the creation and management of custom Web sites and servers." The Defendants assert that this clearly demonstrates that the purpose of the invention was to partition the various modules on separate machines. As to the passages cited by the Plaintiff, the Defendants assert those passages do not describe the entirety of the claimed invention. The Plaintiff asserts that the passage cited by the Defendants is directed toward the Web

site management "need" recited in the passage, and this need is addressed by a partitioned architecture.

The parties have thus each pointed to somewhat conflicting passages of the specification to support their positions. The passages cited by the Plaintiff establish that there is not a clear disavowal within the specification of the use of a partitioned software architecture on a single machine. The Defendants do correctly point to cases which stand for the proposition that when the specification makes clear that the invention does not include a particular feature than that feature is deemed to be outside of the reach of the claims of the patent. Defendants' Joint Sur-Reply, p. 8. However, in the specification before this Court, the specification does not make clear that the invention must only be operated on separate machines.

The Court construes "page server" to be "page-generating software that generates a dynamic Web page."

4. "Web Server"

"Web Server" is utilized in asserted claims 1 and 11 of the '554 Patent and 1-2 of the '335 Patent. The Plaintiff asserts that the proper construction of "Web server" is "a processing system capable of processing an HTTP request and producing a response to such a request." The Defendants assert that the terms should be construed as "a machine running a Web server executable capable of storing, locating, and returning Web pages in response to Web client requests." In the claim construction Oral Argument, the Plaintiff stated that it would agree to language including "software" in place of the Plaintiff's originally proposed "system" language similar to the Plaintiff's agreement with regard to "page server."

The focus point of the dispute between the parties is whether the term "Web server" requires a machine or whether the term may merely represent software or a combination of the two. Both the Plaintiff and Defendants cite conflicting extrinsic evidence to support their positions in the form of dictionaries, industry guides, and protocols. Some of the Plaintiff's extrinsic evidence includes citations to extrinsic evidence first brought before the Court by the Defendants. The conflicting extrinsic evidence presented by the parties fits the rationale presented in *Phillips* regarding the cautions that should be considered relating to such evidence.

Looking to the specification, the Plaintiff points to passages in which "Web server" is not used to describe a machine. In particular, the Plaintiff points to the statement in the specification that:

> The preferred embodiment of the present invention is implemented as a software module, which may be executed on a computer system such as computer system 100 in a conventional manner. Col. 3:55-58.

The Plaintiff also highlights the following passage:

> This embodiment is appropriate for Web servers such as Netsite™ from Netscape, that support such extensions. A number of public domain Web servers, such as NCSA™ from the National Center for Supercomputing Applications at the University of Illinois, Urbana-Champaign, however, do not provide support for this type of extension. Thus, in an alternate embodiment, Interceptor 400 is an independent module, connected via an 'intermediate program' to Web server 201. This intermediate program can be a simple CGI application program that connects Interceptor 400 to Web server 201. Alternate intermediate programs the perform the same functionality can also be implemented. Col. 4:63-Col. 5:7.

The Defendants counter that the specification uses the terms "Web server," "Web server executable" and "Web server machine" and that the proper interpretation is that the machine is referred to as a Web server machine, the software is referred to as the "Web server executable" and

the combination is referred to as a "Web server." To support this argument, the Defendants cite to various passages and figures in the specification. Figures 2-4; Col. 4:39-41; Col. 4:59-62; Col. 5:7-36.

While the Defendants may be correct that "Web server" may be utilized at times as indicating a combination of a machine and software, the specification clearly does not require the term "Web server" to include a machine. The passages of the specification noted above by the Plaintiff make clear that the Web server is contemplated to be at least in one embodiment, software. It is also noted that in general in other passages of the specification the term "Web server machine" is more often used when describing the machine component and "Web server" to describe merely the software component. Thus, for example, it is noted that the "Web servers process each of these requests on a single machine, namely the Web server machine," "Interceptor 400 resides on the Web server machine as an extension to Web server 201," and the Dispatcher "can, however, also reside on the same machine as the Web server." Col. 4:39-42; Col. 4:61-62; Col. 5:20-21. Passages such as these imply a utilization of the term "Web server" as the software module as opposed to the combination of both the machine and software. Thus, some usage of "Web server" implies just software and at other times implies a combination of software and hardware.

The Court construes "Web server" to be "software, or a machine having software, that receives Web page requests and returns Web pages in response to the requests.

5. "HTTP-complaint device"

"HTTP-complaint device" is utilized in asserted claims 15-16 and 19 of the '335 Patent. The Plaintiff asserts that "HTTP-complaint device" does not need construction. Alternatively, if

construed, the Plaintiff asserts that the proper construction is "a device that understands HTTP and whose behavior is affected by an HTTP request." The Defendants assert that the terms should be construed as "a machine running an executable capable of storing, locating and returning Web pages in response to Web client requests."

The Defendants assert the same construction for the terms "Web server" and "HTTP-complaint device." The Defendants' construction is based upon their assertion that the term does not appear in the specification and that the specification only discloses a Web server for performing the function described in the language surrounding the use of "HTTP-compliant device." As such, the Defendants assert that "HTTP-compliant device" should be construed the same as "Web server" as that is the only corresponding device described and enabled in the specification. To hold otherwise, assert the Defendants, would result in a claim that is overbroad and invalid for not being described and enabled.

Regarding the maxim that claims should be construed to be valid, in *Phillips* the Federal Circuit guidance states that this maxim is limited "to cases in which 'the court concludes, after applying all the available tools of claim construction, that the claim is still ambiguous.'" *Phillips*, 415 F.3d at 1327. The Court does not find that in light of the specification the term in question is ambiguous to one skilled in the art. Thus, the Defendants' validity concerns should be more properly addressed with regard to validity motions.

The Defendants also assert that the more general construction proposed by the Plaintiff would be so broad as to even encompass a Web client. However, the claim language itself makes clear that this concern is not valid as there is substantial functional language regarding what happens at the HTTP-compliant device including the transferring of a request from the HTTP-compliant device to

a page server, intercepting the request at the HTTP-compliant device, and concurrently processing other requests at the HTTP-compliant device.

The Defendants do however raise valid concerns over the Plaintiff's definition raising additional interpretation questions with regard to the meaning of "understands HTTP" and "behavior affected by an HTTP request." The Court agrees with the Defendants in this regard. The specification defines HTTP as "a communications protocol known as HyperText Transport Protocol (HTTP)." Col. 1:25-26. Mindful that not all terms in a claim need construction, **the Court adopts a construction of "HTTP-compliant device" to mean "a device that is compliant with the communication protocol known as HyperText Transport Protocol (HTTP)."**

6. "Said processing being performed by said page server while said Web server concurrently processes said other requests"

"Said processing being performed by said page server while said Web server concurrently processes said other requests" is utilized in asserted claims 1 and 11 of the '554 Patent and 1, 15 and 29 of the '335 Patent. The Plaintiff asserts that the "said processing…" phrase does not need construction. Alternatively, if construed, the Plaintiff asserts that the proper construction is "said processing being performed by said page server while said Web server processes said other requests at the same time." The Defendants assert that the phrase should be construed as "said processing being performed by said page server while said Web server executable processes other requests literally at the same time on a different machine."

Three main points of distinction exist between the parties: the inclusion of "executable" with regard to the use of Web server, the inclusion of "literally" with regard to the "same time" language,

and the inclusion of the concept of the Page server and Web servers being on different machines. With regard to the term "Web server" as utilized within the "said processing…" phrase, the Court finds that the construction the Court provided above for "Web server" is applicable, and thus the inclusion of the term executable does not need to be re-addressed with relation to the "said processing…" phrase. Similarly, with regard to the concept of different machines, the Court has addressed that concept above and does not need to re-address that concept here.

With regard to "concurrently," both parties agree that this term includes the concept of something occurring "at the same time;" however, there is still a dispute as to whether it must be "literally at the same time." The Defendants argue that "concurrently" should be analyzed in the context of the discussion in the patents of the prior art time-interleaved multi-threading techniques. Accordingly, the Defendants argue that if concurrently is not read to be "literally at the same time" the claims would read on time-interleaved multi-threading techniques. Once again, such arguments are more suited for invalidity assertions. Moreover, it is noted that in the passage in which the patents utilize the term "concurrently" to describe the Web server and the Page server operations, the patents also describe this concept as "to simultaneously process." Col. 6:21-27. With regard to the prior art time-interleaved multi-threading discussion (which the Defendants assert is not literally at the same time) the patent also uses the term "simultaneously." Col. 4:48-51. The patents do not distinguish one use of the term simultaneously from the other by the inclusion of the literal concept. The Court does not find support in the intrinsic evidence to support a requirement that "at the same time" must be "literally at the same time." **Thus, the court construes "said processing being performed by said page server while said Web server concurrently processes said other**

requests" to mean "said processing being performed by said page server while said Web server processes said other requests at the same time."

7. "Intercepting"

"Intercepting" is utilized in asserted claims 1, 9, 10 and 11 of the '554 Patent and 1 and 15 of the '335 Patent. The Plaintiff asserts that the proper construction of "intercepting" is "stopping, deflecting, or interrupting the processing of a request." The Defendants assert that the term should be construed as "diverting a request received at the Web server machine instead of the Web server executable processing it." Both parties argue that the other party's interpretations carry implicit meanings beyond the mere constructions asserted above. The Plaintiff asserts that the Defendants' construction implicitly requires a request to go around or bypass a Web server without any processing by the Web server. The Defendants assert that the Plaintiff's construction allows for the Web server executable to begin processing a request, even if only to recognize that is should not complete the processing. The parties also disagree as to whether the terms "stopping, deflecting, or interrupting" verses "diverting" are more appropriate. A portion of the disagreement between the parties is based upon the fundamental dispute as to whether a Web server is a machine, software, or combination thereof. As noted above, the Court construes a Web server as software, or a machine having software.

The Defendants assert that the prosecution histories of the '554 Patent and of the '335 Patent add clarity to the meaning of the term "intercepting." The Defendants are correct that prosecution history may be used to limit the claims so as to exclude interpretations that may have been disclaimed or disavowed. However as discussed below, the cited portions of the prosecution

histories of the '554 Patent and the '335 Patent do not amount to a clear disclaimer as suggested by the Defendants. *See Middleton, Inc. v. Minnesota Mining & Mfg. Co.*, 311 F.3d 1384, 1388 (Fed. Cir. 2002).

With regard to the '554 Patent file history, the Defendants point to language added via an Examiner's Amendment and to the Bookman reference for support of a disclaimer of claim scope. The amendment in question was made in a Notice of Allowability that was issued along with an Interview Summary of a December 17, 1998 examiner interview and a form PTO-892 Notice of References Cited. The PTO-892 Notice of References Cited listed the Bookman reference and an additional reference. At that time, other art was also included in the prosecution history including, for example, references that were the basis of previous rejections. The Interview Summary merely states that the prior art discussed was the "prior art of record." The description and other comments of the interview do not provide any other details as to why the Examiner Amendment was made. The Amendment in question added, to claim 1 for example, the language "wherein said routing step further includes the steps of intercepting said request at said Web server, routing said request from said Web server to a dispatcher, and dispatching said request to said page server." As the Plaintiff points out, the record does not show that Bookman was even particularly discussed in the interview let alone the art requiring the amendment to be made. On this basis alone, the Court may reject the Defendants' assertion that because of Bookman the claim language must be interpreted in the manner that the Defendants allege. Further, even if Bookman had been the art requiring such amendment, the record still does not provide any insight into the meaning of "intercepting" as the record is silent as to any further meaning of "intercepting" or the other additional terms recited in the amendment (routing from a Web server to a dispatcher and dispatching said request to a page server).

With regard to the '335 Patent prosecution history, the Defendants assert that statements made by the Applicants regarding the Leaf reference support the Defendants' proposition that partial processing does not equate to intercepting. In particular, the Defendants cite to a quote on pages 9-10 of a Response To Office Action dated May 23, 2001. The Defendants focus on a statement that Leaf did not suggest intercepting because "merely routing a request from a web server to the transaction gateway does not involve interception." The Defendants state that Leaf has Web server executable that partially processes a request before routing it on to the dispatcher. From this, the Defendants argue that the Applicants disclaimed partially processing a request and then routing it to the dispatcher. It is noted that the Applicants' Response in question, however, does not characterize Leaf in the manner suggested by the Defendants or make any references to partially processing. Further, the distinction between a Web server and Web server executable made by the Defendants is not clear in Leaf. In addition, it is noted that the full context of the Applicants' remarks regarding Leaf includes the statement that "Leaf does not teach or suggest 'intercepting said request.' Instead, Leaf teaches that the web server routes the request directly to the transaction gateway client." '335 Patent File History, Response to Office Action Dated May 23, 2001, p. 9-10. This statement merely suggests that directly routing a request from a web server to a transaction gateway is not intercepting and does not provide clear guidance as to the question of partial processing or the difference between a Web server and Web server executable. Thus, the prosecution history cited by the Defendants does not provide the clear guidance asserted by the Defendants.

It is noted that the Defendants' proposed construction adds Web server machine to the term of "intercepting." This language is somewhat redundant with the language surrounding the term

"intercepting" in most claims and does not conform to claim 15 of the '335 Patent which uses the term HTTP-complaint device. The context of the term as used in the claims themselves provides some guidance as to the proper construction. The claims themselves note that the intercepting of the request is at the Web server or the HTTP-compliant device.[3] For example, Claim 1 uses language such as "routing said request from said Web server to a page server," and "wherein said routing step further includes the steps of intercepting said request at said Web server, routing said request from said Web server to a dispatcher, and dispatching said request to said page server." This conforms with the specification, which states that the request is initially routed from the Web Client 200 to the Web Server 201. Col. 4:54-60. Also, Figure 4 appears to show the request going to the Web server executable 201(E). A construction that has the request bypassing the Web server is therefore not appropriate.

As to whether the beginning phrase of the construction should include "stopping, deflecting, or interrupting" as proposed by the Plaintiff, the Plaintiff provides little support for such language other than a general purpose dictionary. Although the Plaintiff asserts that the Defendants' construction requires bypassing the Web server, the Court does not interpret the phrase "diverting" to require such bypassing. However, the term "diverting" seems to carry an additional connotation that the request is sent somewhere else. When looking at the claims, however, the concept of the request being sent elsewhere is included in the "routing said request" (claims 1, 9, and 11 of the '554 Patent) or "transferring said request" (claim 15 of the '335 Patent) limitation that immediately follows the intercepting phrase. Thus, the Court does not feel that either construction adds clarity

[3] Claim 15 of the '335 Patent uses the term HTTP-compliant device while the other claims recite a Web server.

to the meaning of the concept of "intercepting" as used in the claims. Moreover, the parties have not pointed to anything in the intrinsic record that suggests which is more accurate: "diverting" or "stopping, deflecting or interrupting." These terms are not used in the specification and each in turn may need their own construction. The Court is not convinced that the term "intercepting" needs construction itself or that the constructions proposed by the parties add any needed clarity.

More helpful would be to construe the entire intercepting phrase: "intercepting said request at said Web server" (claims 1, 9, and 11 of the '554 Patent) and "intercepting said request at said HTTP-compliant device" (claim 15 of the '554 Patent). The specification describes the Interceptor as intercepting "the handling of a request." Col. 8:31-32. To conform with the description provided within the specification, the phrase "intercepting said request at said Web server" (claims 1, 9, and 11 of the '554 Patent) means at least "intercepting the handling of a request at a Web server" and the phrase "intercepting said request at said HTTP-compliant device" means at least "intercepting the handling of a request at a said HTTP-compliant device."

What is left for the Court to determine is whether the phrase "instead of the Web Server executable processing it" should be added to the end of the definition as proposed by the Defendants. The Defendants' primary support for their position is the language of the specification that states "instead of Web server executable 201(E) processing the URL request, however, interceptor 400 intercepts the request and routes it to Dispatcher 402." Col. 4:58-60. The Defendants argue that this language provides no room for partial processing of a request by Web server executable. The Court, however, finds such language ambiguous. The Defendants would like to interpret this cited quote ("instead of the Web server executable 201(E) processing the URL request") to mean "instead of the Web server executable 201(E) processing **any of** the URL request" wherein the Plaintiff would like

to interpret this citation to mean "instead of the Web server executable 201(E) **completely** processing the URL request" The specification, however, provides little guidance.

As discussed above, however, the specification does establish that a Web server may be software. Further, the specification establishes that the Web server does perform at least some action with relation to a request, namely receiving a request. Col. 4:55-58; Col. 8:28-31; Figure 4; Figure 5. Further, it is noted that in at least one embodiment the interceptor 400 is "an extension of the Web server 201" and the interceptor 400 also performs actions on a request. Col. 4:59-62. As Defendants have asserted that their proposed claim langue would exclude the Web server from any processing of the request, their proposed claim language would impermissibly exclude the embodiments disclosed within the specification. Thus, the Court declines to add the additional limitation sought by the Defendants.

For these reasons, **the Court finds that "intercepting said request at said Web server" means "intercepting the handling of a request at a Web server" and the phrase "intercepting said request at said HTTP-compliant device" means at least "intercepting the handling of a request at a said HTTP-compliant device."**

8. "Transferring"

"Transferring" is utilized in asserted claims 15-16 and 29 of the '335 Patent. The Plaintiff asserts that the "transferring" does not need construction. Alternatively, if construed, the Plaintiff asserts that the proper construction is "sending." The Defendants assert that the term should be construed as "sending toward a destination and relinquishing control of."

The parties agree to the use of the term of "sending" but disagree as to whether any additional

language is necessary. With regard to the Defendants' use of "toward a destination," it is noted that the surrounding claim language of each claim already includes this concept. For example, claim 15 states that the transferring is of "a request from an HTTP-complaint device to a page server." Similarly, claim 29 states that the transferring is of "a request from an HTTP-complaint device to a dispatcher." The inclusion of "toward a destination" within the term "transferring" itself is therefore unnecessary based upon the context of the claims themselves.

With regard to the "relinquishing control of" language sought by the Defendants, the Defendants point to a Microsoft Press Computer Dictionary definition that includes in transferring the concept of passing program control. Further, the Defendants assert that the purpose of the patent is to reduce processing burden. The Plaintiff argues that two other dictionary definitions noted by the Defendants do not include the relinquishing concept.

Once again, it is more instructive to look to the claims themselves. More particularly, in claim 15 of the '335 Patent immediately after the "transferring…to a page server" the claim continues with "said page server receiving said request and releasing said HTTP-compliant device to process other requests…." Likewise, claim 29 of the '335 Patent includes later in the claim "said page server receiving said request and releasing said HTTP-compliant device to process other requests…" To include relinquishing within the definition of transferring would possibly conflict with the latter claim language or alternatively be redundant. Further, to the extent that the Defendants assert that the purpose of the patent is to reduce processing burden, the recited releasing language is more related to that concept then the transferring.

As both parties have proposed definitions using the term "sending," the Court includes that term in its construction. **The Court construes "transferring" to mean "sending."**

9. "Dispatching"

The parties originally proposed different constructions for the term "dispatching." As the parties have subsequently submitted an agreed construction, a Court construction is no longer required.

10. "Releasing"

"Releasing" is utilized in asserted claims 1, 9 and 11 of the '554 Patent and 1, 15, and 29 of the '335 Patent. The Plaintiff asserts that the "releasing" does not need construction. Alternatively, if construed, the Plaintiff asserts that the proper construction is "freeing." The Defendants assert that the term should be construed as "communicating to said Web server that it may now process other requests."[4] A central point in the dispute between the parties is the Plaintiff's arguments that releasing can implicitly occur as a result of routing without communication to the Web server. The Defendants assert that there must be communication to the Web server.

The Plaintiff asserts that the term "freeing" is supported by the specification and notes that the specification states that the result of routing is that the Web server is free to continue servicing client requests. In particular, the Plaintiff points out that the specification states "Web server executable 201(E) is thus free to continue servicing client requests on Web serer 201 while the request is processed 'off-line.'" Col. 5:16-18. Thus, the Plaintiff asserts that the specification implies that releasing is an automatic consequence of routing the request to another processing element and that nothing in the specification requires communication to the Web server to effectuate

[4]For claim 9 of the '554 Patent, the Defendants substitute "second computer system" in place of "Web server" and similarly in claims 15 and 29 of '335 Patent the Defendants substitute "HTTP-compliant device" in place of "Web server."

the release.

The Defendants assert that the '335 Patent prosecution history shows that merely routing a request from a Web server to a Page server and thereby implicitly releasing the Web server was disclaimed by the Applicants during the prosecution history as being different from releasing. The Plaintiff counters by asserting that the full context of the prosecution history quote in question does not stand for the proposition asserted by the Defendants. The portion of the prosecution history in question includes:

> At no time does Rogers teach or suggest 'concurrently' processing other requests or 'releasing said Web server to process other requests' because merely retrieving data from multiple sources does not teach or suggest these elements. Response to Office Action, Nov. 27, 2001 at 9.

The Court is persuaded that by reviewing the full context of the portion of the prosecution history in question a clear disavowal was not made by the Applicants. Thus, the prosecution history does not mandate that releasing cannot implicitly occur due to routing from a web server to a page server.

However, the Defendants raise a more relevant argument with regard to the claim language itself. "Releasing" is used in each claim as part of the phrase "said page server receiving said request and releasing said Web server to process other requests."[5] It is therefore the Page server that does the releasing. The larger context of the use of releasing in the claims themselves indicates that the Page server has a role in the releasing as in the claims themselves it is the Page server that releases the Web server. This language also conforms to the Summary of Invention and Abstract. Col. 2:25-26; Abstract, line 8.

[5] "Second computer system" is used rather than "Web server" in claim 9 of the '554 Patent. "HTTP-compliant device" is used rather than "Web server" in claims 15 and 29 of '335 Patent.

The Court is thus persuaded that the Defendants are partly correct that as required by the claim language itself the Page server takes some action to releasing the Web server. However, the Defendants do not provide adequate support in the specification or elsewhere to mandate that such action is limited to communication from the Page server to the Web server. As there could be other actions that the Page server may take to affirmatively release the Web server, it would be inappropriate to limit the claim to one type of action, particularly when support is lacking in the record for that particular type of action. The specification does not necessarily limit the claims to a particular technique by the Page server as to how the claimed release by the Page server is accomplished. For example, the limitations proposed by Defendants could be argued to not include Page server actions that are communications from the Page server to other intermediate elements (such as the Dispatcher) or Page server actions that involve affirmatively not sending some expected regular communication. **The Court therefore interprets "said page server receiving said request and releasing said Web server to process other requests" to mean "said page server receiving said request and said page server performing an act (separate from merely receiving the request) to free the Web server to process other requests." Claims 9 of the '554 Patent and claims 15 and 29 of the '335 Patent are likewise construed with the substitution of "second computer system" and "HTTP-compliant device" respectively for "Web server."**

B. Means Plus Function Elements

The parties propose different constructions of three means plus function elements. In each case, the parties agree to the function but disagree as to what is the corresponding structure that is

disclosed in the specification pursuant to 35 U.S.C. §112, ¶6. The following means plus function elements contained in claim 9 of the '554 Patent are disputed:

> (a) "means for generating said request,"
> (b) "means for receiving said request from said first computer" and
> (c)"page server processing means processing said requests and dynamically generating a web page in response to said request."

Throughout the pre-hearing and post-hearing briefing, the parties have substantially changed their positions as to the appropriate legal standards and the appropriate structure that should be applied to each element.

As to the function, the parties have agreed to functions that match the claimed language recited above: "generating said request," "receiving said request from said first computer," and "processing said requests and dynamically generating a web page in response to said request" respectively.

The Plaintiffs propose that the corresponding structure for each means plus function element is "any software, processor or equivalent therefore that performs the function of _____" (where the blank is the agreed function for that means plus function element). The Defendants in contrast point to particular structure disclosed within the specification. For the "means for generating," the Defendants propose "a processor, such as Box 102 of Figure 1, on a Web client 200 of Figure 4, running a Web browser." For the "means for receiving," the Defendants propose "Web server 201 of Figure 4 running a web server executable." For the "page server processing means," the Defendants propose "a processor, such as Box 102 of Figure 1, on a page server machine, such as Box 404(1)-(n) of Figure 4, running undisclosed page-generating software."

First, the Court notes that means plus function elements shall be construed to cover the corresponding structure described in the specification and equivalents thereof. 35 U.S.C. §112, ¶6. To avoid confusion, the Plaintiff indicated that it did not object to removing the word "equivalents" internal to its proposed construction. Plaintiff's Reply Brief at 58. The Court agrees that it would be clearer for "equivalents" to modify the entire construed structure.

As proposed by the Plaintiff, the corresponding structure could be any software or could be any processor that performs the agreed function. As proposed by the Plaintiff, software would not even be required as the Plaintiff uses the "or" conjunction. Plaintiff's construction of any software or processor is essentially unbounded and contradicts the guidance that has been established for interpreting means plus function elements. *Medical Instrumentation and Diagnostics Corp. v. Elekta AB*, 344 F.3d, 1205, 1211 (Fed. Cir. 2003)("We cannot allow a patentee to claim in function terms essentially unbounded by any reference to what one skilled in the art would understand from the public record.") Having chosen to utilize the means plus function claiming structure, the Plaintiff cannot now avoid its implications. Moreover, for computer implemented implementations the corresponding structure for means elements is limited to the specific structure and specific algorithms disclosed in the specification as opposed to the generic "software" or "processor" as asserted by the Plaintiff. *See Harris Corp. v. Ericsson Inc.*, 417 F.3d 1241, 1253-54 (Fed. Cir. 2005); *WMS Gaming Inc. v. International Game Technology*, 184 F.3d 1339, 1348-49 (Fed. Cir. 1999). Thus, the court must look to the specific structures and algorithms disclosed with the specification.

As noted above, the specification states:

> FIG. 1 illustrates a typical computer system 100 in which the present invention operates. Col. 2:66-67.
>
> The preferred embodiment of the present invention is implemented as a software module, which may be executed on a computer system such as computer system 100 in a conventional manner. Col. 3:55-58.

As described within the specification, the various software components operate on computer systems such as computer system 100 of Figure 1 that includes a processor 102.[6] Col. 2:66-67; Col. 3:8-11; Col. 3:55-63.

1. "means for generating said requests,"

For example, the "means for generating a request" is described as a Web client machine running a Web browser. Col. 1:24-26; Col. 2 :4-7; Col. 4:12-15; Col. 4:55-57; Col. 6:27-31; *See* Col. 8:25-29. This particular structure is noted in the specification for accomplishing the claimed function. One skilled in the art would understand from the specification that a Web client computer having a processor which operates a Web browser is the corresponding structure that accomplishes the function of generating a request.

The Court construes the corresponding structure of the "means for generating" to be "a processor of a computer that is, or has, a Web client running a Web browser" or equivalents thereof.

[6] It is noted that as to the component of the computer system, both parties agree that reference to the processor is appropriate.

2. "means for receiving said request from said first computer"

With regard to the "means for receiving said request from said first computer" element, the specification shows the Web server 201 receiving requests from the Web client 200. Figure 4; Figure 5; Col. 4:54-59; *See* Figure 2; Col. 3:64-Col. 4:10. The Web server 201 is also repeatedly described as including Web server executable. *Id.* As described above, the specification also establishes that that even if a Web server is software, the software module operates on a computer as described within the specification.

The Court construes the corresponding structure of the "means for receiving" to be "a processor of a computer that is, or has, a Web server running Web server executable" or equivalents thereof.

3. "page server processing means processing said requests and dynamically generating a web page in response to said request."

With regard to the function of the "page server processing means processing said requests and dynamically generating a web page in response to said request" element, the specification shows that such function is accomplished by Page server 404(1)-(n). Figure 4; Figure 5; Col. 5:37-Col. 6:31; Col. 8:39-43. As noted above, the parties agree that a Page server is page generation software. Further, as also noted above with the other means plus function elements, the specification establishes that software is operated on a computer system as described with the specification. Moreover, with regard to Page servers, it is noted that in one embodiment Page servers reside on a separate machine to accomplish the claimed function. Col. 5 line 49-50. Thus, the Court finds that

the Page server processing means is Page server software (as construed above) operating on a computer processor.

The Court construes the corresponding structure of the "page server processing means" to be "a processor of a computer that runs Page server software (wherein Page server software is page-generating software that generates a dynamic Web page)."

IT IS SO RECOMMENDED.

Within ten (10) days after receipt of the magistrate judge's report, any party may serve and file written objections to the findings and recommendations of the magistrate judge. 28 U.S.C.A. 636(b)(1)(C).

Failure to file written objections to the proposed findings and recommendations contained in this report within ten days after service shall bar an aggrieved party from *de novo* review by the district court of the proposed findings and recommendations and from appellate review of factual findings accepted or adopted by the district court except on grounds of plain error or manifest injustice. *Thomas v. Arn*, 474 U.S. 140, 148 (1985); *Rodriguez v. Bowen*, 857 F.2d 275, 276-77 (5th Cir. 1988).

SIGNED this 15th day of August, 2006.

CAROLINE M. CRAVEN
UNITED STATES MAGISTRATE JUDGE

A34

IN THE UNITED STATES DISTRICT COURT
OF THE EASTERN DISTRICT OF TEXAS
TEXARKANA DIVISION

EPICREALM, LICENSING, LLC	§	
	§	
v	§	**No. 5:07CV125**
	§	
AUTOFLEX LEASING, INC., et al.	§	

EPICREALM, LICENSING, LLC	§	
	§	
v	§	**No. 5:07CV126**
	§	
FRANKLIN COVEY CO., et al.	§	

EPICREALM, LICENSING, LLC	§	
	§	
v	§	**No. 5:07CV135**
	§	
VARIOUS, INC.	§	

ORDER

Pursuant to the provisions of 28 U.S.C. § 636(b)(1) and (3) and the Amended Order for the Adoption of Local Rules for Assignment of Duties to United States Magistrate Judges, the above-entitled and numbered cause of action was referred to the undersigned for pretrial purposes. Before the Court is Plaintiff epicRealm Licensing, LP's Motion to Substitute Parties Pursuant to FED. R. CIV. P. 25(c)(Docket Entry #s 440, 409, 73). The Court, having reviewed the relevant briefing and hearing arguments of counsel June 24, 2008, is of the opinion the motion should be **DENIED** at this time. Pursuant to Rule 25(c), Parallel Networks, LLC is joined as a party plaintiff and counter-defendant with epicRealm Licensing, LP in this action.

I.

PLAINTIFF'S MOTION

[E]picRealm Licensing, LP ("Plaintiff" or "epicRealm") moves, pursuant to Federal Rule of Civil Procedure 25(c), to substitute Parallel Networks, LLC ("Parallel Networks") for epicRealm as the plaintiff and counter-defendant in this action. In August of 2007, epicRealm assigned to Parallel Networks all rights, title, and interest in the two patents in suit, United States Patent No. 5,894,554 ("the '554 patent") and United States Patent No. 6,415,335 ("the '335 patent")(collectively, the "patents in suit"), including the right to enforce the patents in suit and to recover for past infringement. Plaintiff argues it has no interest in the patents in suit, and Parallel Networks should be substituted as the sole plaintiff and counter-defendant in this action. Plaintiff asserts substitution, rather than joinder, is the preferred procedure over a motion to join for adding a necessary party.

Plaintiff further asserts substitution of Parallel Networks for epicRealm will cure any jurisdictional issue that may be argued by the failure of epicRealm as the named plaintiff and counter-defendant to have any interest in the patents in suit and by the failure to name the true owner of the patents in suit as a party to this action. Plaintiff emphasizes it was dissolved and ceased to exist as an entity as of October 31, 2007, leaving Parallel Networks the owner of the entire interest in the patents in suit.

Defendants have no objection to joining Parallel Networks as a party to this case. However, in lieu of substitution, Defendants request the Court exercise the discretion given it by FED. R. CIV. P. 25(c) and keep epicRealm a party to the case. Specifically, Defendants assert there is a serious and substantial question regarding epicRealm's ownership interests in the patents-in-suit. Defendants rely on the corporate and patent history of epicRealm to question the scope of

epicRealm's, and hence Parallel Networks' interest in the patents which form the basis for this infringement action. The Court summarizes Defendants' challenge below.

On April 13, 1999, the '554 Patent was issued and assigned to InfoSpinner, Inc. On January 31, 2000, InfoSpinner, Inc. changed its name to epicRealm, Inc. (On December 31, 2000, epicRealm, Inc. merged with another corporate entity to form epicRealm Operating, Inc., which thereby became the owner of the '554 Patent. On July 2, 2002, the '335 Patent was issued and assigned to epicRealm Operating, Inc. On December 5, 2003 the dissolution of epicRealm Operating, Inc. was authorized by its shareholders, and a Certificate of Dissolution was filed in Delaware on January 20, 2004. At the time of these events, it appears that epicRealm Operating, Inc. owned both patents.

On March 4, 2005, epicRealm, Inc., which had been dissolved along with epicRealm Operating, Inc. on January 20, 2004, purported to sell the '554 and '335 Patents to epicRealm Licensing, LLC, a limited liability company formed in Texas. This agreement did not purport to specifically convey to epicRealm Licensing, LLC the right to recover for past infringements, and epicRealm, Inc. retained the right to share in any litigation or enforcement proceeds realized post-agreement.

According to Defendants, the effect of this transaction is further called into question by activity which occurred a month later. On April 8, 2005, epicRealm Operating, Inc., not epicRealm, Inc., filed with the U.S. Patent and Trademark Office a notice of a *nunc pro tunc* assignment of the patents at issue (along with its other intellectual property) effective retroactively to March 4, 2005 to epicRealm Licensing, LLC. The instant suits were initially brought by epicRealm Licensing, LLC. The current plaintiff, epicRealm Licensing LP, is the successor to epicRealm Licensing, LLC via merger.

Defendants argue that if epicRealm Operating, Inc. owned the patents, and absent some other agreement or the legal effect of Delaware statutory provisions, none of which are apparent to Defendants at this time, epicRealm, Inc. did not and could not convey any title to those patents. Defendants state they will soon be filing a separate motion regarding this standing issue.

II.

APPLICABLE LAW

Federal Rule of Civil Procedure 25 governs the procedure for substituting parties in certain cases, specifically, where there has been a death, incompetency of a party, or transfer of interest. Rule 25(c) provides:

> **Transfer of Interest.** In case of any transfer of interest, the action may be continued by or against the original party, unless the court upon motion directs the person to whom the interest is transferred to be substituted in the action or joined with the original party. Service of the motion shall be made as provided in subdivision (a) of this rule.

FED. R. CIV. P. 25(c).

A transfer of the interest in an action occurs when, for example, "one corporation becomes the successor to another by merger or other acquisition of the interest the original party had in the lawsuit," *Luxliner P.L. Exp., Co. v. RDI/Luxliner, Inc.*, 13 F.3d 69, 72 (3d Cir. 1993), or when the new party is the transferee of the patent in suit. *Hazeltine Corp. v. Kirkpatrick*, 165 F.2d 683, 685 (3d Cir. 1948). Rule 25(c) allows a court to join a successor to the interest of a party with, or substitute it for, the predecessor to the interest in a lawsuit. *Luxliner*, 13 F.3d at 70.

Rule 25(c) does not require a party to take any action after an interest has been transferred. *Id.* at 71. "When a defendant corporation has merged with another corporation, for example, the case may be continued against the original defendant and the judgment will be binding on the successor

even if the successor is not named in the lawsuit." *Id.* Since joinder or substitution under Rule 25(c) is a procedural device that does not typically alter the substantive rights of a party, a Rule 25(c) decision is generally left to the court's discretion. *Id.* at 71-72. "To determine whether an entity is a transferee of interest so as to trigger this discretion, however, a district court's mission is one of applying law to facts." *Id.* at 72.

The Court could not locate any instructive authority from the Fifth Circuit Court of Appeals on the procedure for determining a Rule 25(c) motion. However, the Third Circuit Court of Appeals has considered in detail Rule 25(c). *See Luxliner P.L. Exp., Co. v. RDI/Luxliner, Inc.*, 13 F.3d 69 (3d Cir. 1993) According to the Third Circuit, unlike FED. R. CIV. P. 56 regarding summary judgment, "Rule 25(c) does not specify a method for deciding motions or a standard to use in determining whether motions can be decided on the papers." *Id.* at 72. The "gap" in Rule 25(c) most likely stems from the fact that "the rule does not easily lend itself to contested motions practice; it permits automatic continuation of a lawsuit against an original corporate party, although the outcome will bind the successor corporation, unless the court believes the transferee's presence would facilitate the conduct of the litigation." *Id.* In the absence of such procedures in the rule, the Third Circuit delineated the procedures required to accord due process to the parties. *Id.*

The Third Circuit held that where competing affidavits focus on a genuine issue of material fact, a district court may not decide factual issues arising in the context of a Rule 25(c) motion simply by weighing the affidavits against one another. *Id.* Instead, the court should first determine whether the affidavits "show that there is no genuine issue of material fact and that the moving party is entitled to [joinder or substitution] as a matter of law." *Id.* If they do, the court should grant the motion; if they do not, the court should conduct an evidentiary hearing to decide whether the motion

should be granted. *Id.* at 72-73.

The Court located another case instructive on the issue currently before the Court. In *Mars, Inc. v. JCM American Corp.*, 2007 WL 776786 (D.N.J. 2007)(unreported), the plaintiff moved under Rule 25(c) to substitute MEI, Inc. for Mars, Incorporated ("Mars") as the named plaintiff based on its claim that the rights and interests in the patents-in-suit were transferred from Mars to MEI. *Id.* at *1. The defendants opposed Mars' motion, arguing the assignment of the patent to MEI was not evidence that "Mars actually transferred the entire right, title and interest to the *Reexamination Application which replaces the '352 Patent-in-Suit* over to MEI." *Id.* (emphasis in original). The defendants further argued that the pending counterclaims against Mars for initiating the action "based upon patents which it knew to be invalid and not infringed" prevented complete dismissal of Mars. *Id.* In addition, the defendants argued that MEI was not the proper party to enforce the patents-in-suit because "the assets of MEI were sold to a confidential buyer."[1] *Id.*

The only evidence before the court in *Mars* on the respective rights and interests of Mars and MEI for the patents-in-suit were the competing affidavits of the parties. *Id.* at *2. While the parties agreed that at one time at least a portion of the rights and interests in the patents-in-suit were transferred from Mars to MEI, the parties disagreed "concerning the extent of the rights and interests that presently exist." *Id.*

Because the record provided by the parties did not clearly indicate whether Mars, MEI, or a third party presently owns the rights and interests of the patents-in-suit, the Court declined to rule

[1] Mars asserted it sold all of its rights and interests in the patents-in-suit to MEI and the assets of MEI were sold to a confidential buyer. The defendants asserted the assignment of the reexamination rights under the patents-in-suit were not transferred from Mars to MEI and/or the rights and interests now lie with the confidential buyer, not MEI. *Mars*, 2007 WL 776786 at *2.

at that time that MEI should be substituted for Mars. *Id.* The court relied upon the Third Circuit decision in *Luxliner*, wherein the court considered an analogous situation. According to the court in *Mars*, the "decision in *Luxliner* indicates that a court cannot decide disputed successor liability fact issues by simply evaluating the parties' affidavits." *Id.* Instead, an evidentiary hearing is necessary to resolve the issue. *Id.* Given the uncertainty over the rights and interests in the patents-in-suit in the *Mars* case, the court declined to substitute MEI for Mars. The Court exercised its discretion in joining MEI as a party plaintiff with Mars pursuant to Rule 25(c).[2] *Id.*

III.

DISCUSSION

This action involves three consolidated lawsuits filed by epicRealm, case no. 5:07-CV-125 (filed May 2, 2005), case no. 5:07-CV-126 (August 5, 2005), and case no. 5:07-CV-135 (filed January 25, 2007). At the time the complaint and the counterclaims were filed in each of these lawsuits, epicRealm was the owner of the patents in suit and had the right to enforce the patents against infringers. On August 31, 2007, epicRealm assigned to Parallel Networks all rights, title, and interest in the patents in suit, including the right to enforce the patents and the right to recover for past infringement. [E]picRealm was dissolved and ceased to exist as a legal entity on October 31, 2007. (Plaintiff's mot., Exh. B at ¶3, Exh. C).

[2] The court noted in a footnote that the defendants argued in their surreply that Mars and MEI did not have standing to pursue the litigation. *Mars*, 2007 WL 776786, *2 at fn. 2. Because the plaintiff had not had an adequate opportunity to fully address the defendants' standing objection, the court declined to address the defendants' standing objection at that time. The court ordered the defendants to file a separate motion. *Id.*

Plaintiff relies on three exhibits in support of its motion to substitute: (1) the United States Patent and Trademark Office Notice of Recordation of Assignment Document dated September 10, 2007; (2) the Declaration of Terry Fokas, wherein Fokas declares, among other things, that on August 31, 2007, epicRealm Licensing, LP assigned all rights, title, and interest in and to the patents in suit to Parallel Networks; and (3) the Delaware Certificate of Cancellation In response, Defendants rely on a summary of epicRealm's corporate/patent/filing history.

Because Defendants did not attach a competing declaration or affidavit in support of their briefing, this is not a situation like that in *Luxliner* where the parties submitted competing affidavits concerning the alleged successor liability of two companies, requiring the Court to conduct an evidentiary hearing to resolve the issue in the context of a Rule 25(c) motion. Here, the Court finds, without need of an evidentiary hearing, that the record at this time in insufficient to determine exactly what rights and obligations were transferred from epicRealm Licensing, LP to Parallel Networks. The evidence relied upon by Plaintiff, and Defendants' argument as to the gaps in the evidence, create uncertainty over the rights and interests in the patents in suit.[3]

For example, in its reply, epicRealm relies upon the March 4, 2005 Asset Purchase and Intellectual Property Assignment Agreement (the "Agreement") entered into between epicRealm, Inc. and epicRealm Licensing, LLC (the entity that filed the current lawsuits). At that time, epicRealm Operating, Inc., a wholly-owned subsidiary of epicRealm, Inc., owned the patents in suit. According to epicRealm, the Agreement transferred epicRealm Operating, Inc.'s interests in the patents to epicRealm Licensing LLC because epicRealm, Inc., as the parent, transferred its

[3] Defendants have also represented to the Court they will soon be filing a separate motion regarding the standing issue. This will assure that epicRealm and Parallel Networks have a full and fair opportunity to respond to Defendants' challenges.

subsidiary's property. On page 2 of its Reply, epicRealm defines the parties to the agreement: "('Seller [epicRealm, Inc., as the owner of the epicRealm Operating, Inc.] agrees to sell, assign, transfer. . . to Buyer . . . .')." However, as urged by Defendants, the bracketed material does not appear in the Asset Purchase agreement. What's more, there is no reference in the documents to epicRealm Operating, Inc. On its face, the Asset Purchase agreement is an agreement between epicRealm, Inc. and epicRealm Licensing, LLC.

Given the uncertainty over the rights and interests in the patents in suit, established by Plaintiff's evidence alone, substitution is not appropriate in this instance. The Court exercises its discretion to decline to substitute Parallel Networks for epicRealm Licensing, LP. In an effort to assure that the proper parties are named in this case, the Court joins Parallel Networks as a party plaintiff and counter-defendant with epicRealm Licensing, LP pursuant to Rule 25(c). Accordingly, it is

ORDERED that Plaintiff epicRealm Licensing, LP's Motion to Substitute Parties Pursuant to FED. R. CIV. P. 25(c)(Docket Entry #s 440, 409, 73) is **DENIED** at this time. It is further

ORDERED that pursuant to FED. R. CIV. P. 25(c), Parallel Networks, LLC is joined as a party plaintiff and counter-defendant with epicRealm Licensing, LP in this action.

SIGNED this 26th day of June, 2008.

CAROLINE M. CRAVEN
UNITED STATES MAGISTRATE JUDGE

A35

IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

ORACLE CORPORATION and ORACLE U.S.A. INC.,	)	
Plaintiffs/Counterclaim Defendants,	)	C.A. No. 06-414-SLR
v.	)	**JURY TRIAL DEMANDED**
EPICREALM LICENSING, LP,	)	
Defendant/Counterclaim Plaintiff.	)	

DEFENDANT'S RESPONSE TO PLAINTIFFS' FOURTH SET OF INTERROGATORIES (NOS. 22-25)

Pursuant to Rules 26 and 33 of the Federal Rules of Civil Procedure, Defendant and Counterclaim Plaintiff epicRealm Licensing, LP ("epicRealm" or "Defendant") by its undersigned attorneys, hereby objects and responds to Oracle's Fourth Set of Interrogatories (Nos. 22-25) to Defendant epicRealm Licensing, LP with the answers and objections set forth below. Pursuant to Federal Rule of Civil Procedure 26(e), epicRealm expressly reserves the right to supplement these responses.

GENERAL OBJECTIONS

1. EpicRealm objects to Oracle's interrogatories, instructions, and definitions to the extent the interrogatories seek information that is protected from discovery under the attorney-client privilege or is immune from discovery under the work product doctrine, or that is otherwise not discoverable under the Federal Rules of Civil Procedure or the Local Rules of the District of Delaware.

2. EpicRealm objects to Oracle's interrogatories, instructions and definitions to the extent they are vague, ambiguous and/or overly broad, and/or unduly burdensome and without

reasonable limitation in scope, and to the extent they relate to subject matters not relevant to this case.

3. EpicRealm objects to Oracle's interrogatories to the extent they are premature.

4. EpicRealm objects to Oracle's interrogatories, instructions and definitions to the extent they seek to impose greater burdens on epicRealm than those required by the Federal Rules of Civil Procedure or the Local Rules of the District of Delaware. In response to Oracle's interrogatories, epicRealm responds in accordance with its obligations under the Federal Rules of Civil Procedure and the Local Rules of the District of Delaware.

5. The foregoing General Objections are incorporated in and made a part of epicRealm's response to each interrogatory. Each of epicRealm's responses is made without waiver or prejudice to these General Objections.

DEFINITIONS

Defendant hereby incorporates by reference Definition Nos. 1-8, including subparts, made in its initial interrogatory responses and Definition No. 9 in Defendant's First Supplemental Response to Plaintiffs' Interrogatory Nos. 4 and 12.

ANSWERS AND OBJECTIONS TO INDIVIDUAL INTERROGATORIES

EpicRealm expressly incorporates the above General Objections as though set forth fully in each response to each of the following individual interrogatories, and, to the extent that those objections are not raised in any particular response, epicRealm does not waive them. An answer to an interrogatory shall not be deemed a waiver of any applicable specific or General Objection to an interrogatory. Furthermore, the specific responses set forth below are based upon information now available to epicRealm, and epicRealm reserves the right, at any time to revise, correct, supplement, add to, or clarify the answers or objections set forth herein.

INTERROGATORY NO. 22: State all facts to support your contention, if any, that epicRealm's damages for Oracle's alleged infringement of the epicRealm patents are not barred in whole or in part by the doctrine of laches, and with respect to each such fact, identify each person with knowledge of that fact and identify any documents that you intend to rely on to support your contention.

ANSWER TO INTERROGATORY NO. 22

Defendant objects to this Interrogatory No. 22 as premature because discovery is ongoing and to the extent that it requests information protected from discovery by the attorney client privilege and/or the work product doctrine.

Subject to and without waiving its General Objections and specific objections, epicRealm answers as follows. EpicRealm contends that Oracle will be unable to prove any facts that establish Oracle's right to a laches defense. In Oracle's answer to Interrogatory No. 4, Oracle failed to identify any specific facts or documents supporting its contention that epicRealm knew or should have known of Oracle's infringement at a time that would give rise to unreasonable delay in bringing a patent infringement action against Oracle, nor any specific facts or documents supporting Oracle's contention that it had been prejudiced by any alleged delay by epicRealm. Given Oracle's incomplete interrogatory response, epicRealm served a notice of deposition under Rule 30(b)(6), Topic Nos. 31-37, seeking discovery of any facts that support Oracle's laches contention. Oracle has not yet provided a witness on those issues. EpicRealm did not know, and should not have known, of Oracle's infringement at a time that would give rise to an unreasonable delay in bringing a patent infringement action against Oracle.

EpicRealm reserves the right to supplement this interrogatory as new information becomes available to epicRealm and after completion of the aforementioned Rule 30(b)(6) deposition.

Persons who may have knowledge relating to this issue include Brad Carl, Terry Fokas and Keith Lowery. Documents directed to this issue include those documents produced in this litigation as well as the deposition testimony of Brad Carl and Terry Fokas.

INTERROGATORY NO. 23: For each Oracle product that you contend infringes one or more claims of the epicRealm patents, specifically identify each accused "interceptor," or other component that performs an act of "intercepting [a] request at [a] Web Server," of that product and state in detail how each "interceptor" or such other component "intercepts" requests at a "Web Server" according to each asserted claim of the epicRealm patents.

ANSWER TO INTERROGATORY NO. 23

Defendant objects to Interrogatory No. 23 as premature because discovery is ongoing and because much of the requested information is in Oracle's possession. Specifically, Defendant is unable to provide a complete answer to this Interrogatory because Oracle's witnesses on Defendant's Rule 30(b)(6) Topics 2-9 were not sufficiently prepared to testify as to the Oracle Database. Defendant further objects to this interrogatory to the extent it requests information protected from production by the attorney-client privilege and/or the work product doctrine. Defendant objects to this interrogatory as overly broad and unduly burdensome.

Subject to and without waiving its General Objections and specific objections, Defendant answers as follows:

For the Oracle Application Server, a dynamic Web page generation request is intercepted at the Oracle HTTP Server and is routed to a dispatcher, as set forth in various claims of the epicRealm patents. The dynamic Web page generation request is not fulfilled solely by the Oracle HTTP Server but, instead, is routed to other components.

For the Oracle Web Cache, a dynamic Web page generation request (at least those that are stateless cache misses) is intercepted at the Oracle Web Cache and routed to a dispatcher, as

set forth in various claims of the epicRealm patents. The dynamic Web page generation request is not fulfilled solely by Oracle Web Cache but, instead, is routed to other components.

Defendant will supplement its response to this interrogatory after completing its depositions relating to the Oracle Database.

INTERROGATORY NO. 24: For each Oracle product that you contend infringes one or more claims of the epicRealm patents, specifically identify each accused "page server" of that product and state in detail how each page server "releases" a "Web server," "HTTP-compliant device" or "second computer system" to "process other requests" according to each asserted claim of the epicRealm patents.

RESPONSE TO INTERROGATORY NO. 24:

Defendant objects to Interrogatory No. 24 as premature because discovery is ongoing and because much of the requested information is in Oracle's possession. Specifically, Defendant is unable to provide a complete answer to this Interrogatory because Oracle's witnesses on Defendant's Rule 30(b)(6) Topics 2-9 were not sufficiently prepared to testify as to the Oracle Database. Defendant further objects to this interrogatory to the extent it requests information protected from production by the attorney-client privilege and/or the work product doctrine. Defendant objects to this interrogatory as overly broad and unduly burdensome.

Subject to and without waiving its General Objections and specific objections, Defendant answers as follows:

For the Oracle Application Server, any page-generating software that generates a dynamic Web page can be a page server. A page server can comprise, *e.g.*, OC4J. A page server can release Oracle HTTP Server in at least two ways. First, the page server releases Oracle HTTP Server by processing the dynamic Web page generation request. By processing the dynamic Web page generation request, the page server causes Oracle HTTP Server (including, without limitation, the processing resources available) to be released or freed to process other

dynamic Web page generation requests. Second, the page server releases Oracle HTTP Server when it acknowledges the receipt of data via the underlying network protocol, *e.g.,* TCP/IP.

For the Oracle Web Cache, any page-generating software that generates a dynamic Web page can be a page server. A page server can comprise, *e.g.,* OC4J in conjuction with Oracle HTTP Server. A page server can release Oracle Web Cache in two ways, *i.e.*, the processing of the dynamic Web page generation request by the page server and the acknowledgement of the receipt of data by the underlying network protocol.

Defendant will supplement its response to this interrogatory after completing its depositions relating to the Oracle Database.

INTERROGATORY NO. 25: For each Oracle product that you contend infringes one or more claims of the epicRealm patents, specifically identify each accused "dispatcher" of that product and state in detail how each dispatcher "dispatches" requests to a "page server," including but not limited to stating in detail how each dispatcher "examin[es] a request to make an informed selection of which pager [sic] server should process the request based on dynamic information maintained about page servers, the dynamic information indicating which pager [sic] server can more efficiently process the request, and send[s] the request to the selected pager [sic] server" according to epicRealm's proposed construction of the term "dispatching."

RESPONSE TO INTERROGATORY NO. 25:

Defendant objects to Interrogatory No. 25 as premature because discovery is ongoing and because much of the requested information is in Oracle's possession. Specifically, Defendant is unable to provide a complete answer to this Interrogatory because Oracle's witnesses on Defendant's Rule 30(b)(6) Topics 2-9 were not sufficiently prepared to testify as to the Oracle Database. Defendant further objects to this interrogatory to the extent it requests information protected from production by the attorney-client privilege and/or the work product doctrine. Defendant objects to this interrogatory as overly broad and unduly burdensome.

Subject to and without waiving its General Objections and specific objections, Defendant answers as follows:

For the Oracle Application Server, mod_oc4j, for example, is a "dispatcher" because it can use either of the metric based load balancing policies. In doing so, mod_oc4j examines dynamic Web page generation requests and routes those requests based on dynamic information maintained about page servers, such as dynamic information maintained about OC4J processes.

For the Oracle Web Cache, the Oracle Web Cache dispatches dynamic Web page generation requests (at least those that are stateless cache misses) to the server with the most weighted available capacity. The weighted available capacity is determined based on dynamic information maintained about the servers.

Defendant will supplement its response to this interrogatory after completing its depositions relating to the Oracle Database.

POTTER ANDERSON & CORROON LLP

OF COUNSEL:

Harry J. Roper
George S. Bosy
Aaron A. Barlow
Patrick L. Patras
David R. Bennett
Paul D. Margolis
Benjamin J. Bradford
Angela M. Terry
Emily C. Johnson
JENNER & BLOCK
330 N. Wabash Avenue
Chicago, IL 60611-7603
Tel: (312) 923-8305

By: */s/ David E. Moore*
Richard L. Horwitz (#2246)
David E. Moore (#3983)
Hercules Plaza, 6th Floor
1313 N. Market Street
Wilmington, DE 19899
Tel: (302) 984-6000
rhorwitz@potteranderson.com
dmoore@potteranderson.com

Attorneys for Defendant/Counterclaim Plaintiff epicRealm Licensing, LP

Dated: March 10, 2008
853867 / 31393 / Oracle

CERTIFICATE OF SERVICE

I, David E. Moore, hereby certify that on March 10, 2008, a true and correct copy of the attached document was caused to be served on the attorney of record at the following addresses as indicated:

VIA HAND DELIVERY AND ELECTRONIC MAIL

Mary B. Graham
Morris, Nichols, Arsht & Tunnell LLP
1201 North Market Street
P.O. Box 1347
Wilmington, DE 19899
MGraham@MNAT.com

VIA ELECTRONIC MAIL

James G. Gilliland
Theodore T. Herhold
Robert J. Artuz
Eric M. Hutchins
Eric A. Mercer
Joseph A. Greco
Townsend and Townsend and Crew LLP
379 Lytton Avenue
Palo Alto, CA 94301
OracleEpicrealm@townsend.com

Chad E. King
Townsend and Townsend and Crew LLP
1200 Seventeenth Street
Suite 2700
Denver, CO 80202
OracleEpicrealm@townsend.com

/s/ David E. Moore
David E. Moore

797128 / 31393 / Oracle