SPENCER HOSIE (CA Bar No. 101777)
shosie@hosielaw.com
DIANE S. RICE (CA Bar No. 118303)
drice@hosielaw.com
DARRELL R. ATKINSON (CA Bar No. 280564)
datkinson@hosielaw.com
HOSIE RICE LLP
600 Montgomery Street, 34th Floor
San Francisco, CA 94111
(415) 247-6000 Tel.
(415) 247-6001 Fax

Attorneys for Plaintiff
MASTEROBJECTS, INC.

DMR

UNITED STATES DISTRICT COURT
FOR THE NORTHERN DISTRICT OF CALIFORNIA

CV 13 4304

MASTEROBJECTS, INC., Plaintiff, v. GOOGLE INC., Defendant.	Case No. **COMPLAINT AND DEMAND FOR JURY TRIAL**

Plaintiff MasterObjects, Inc. ("MasterObjects" or "Plaintiff") hereby files its complaint against defendant Google Inc. ("Google" or "Defendant"), for patent infringement. For its complaint, Plaintiff alleges, on personal knowledge as to its own acts and on information and belief as to all other matters, as follows:

PARTIES

1. MasterObjects is a corporation organized under the laws of the State of Delaware, with its principal place of business in San Francisco, California, prior to January 1, 2010, and now Maarssen, Netherlands.

2. Google is a corporation organized under the laws of the State of Delaware, with its principal place of business in Mountain View, California.

JURISDICTION AND VENUE

3. This complaint asserts a cause of action for patent infringement under the Patent Act, 35 U.S.C. § 271. This Court has subject matter jurisdiction over this matter by virtue of 28 U.S.C. § 1338(a). Venue is proper in this Court by virtue of 28 U.S.C. § 1391(b) and (c) and 28 U.S.C. § 1400(b), in that Google may be found in this district, has committed acts of infringement in this district, and a substantial part of the events giving rise to the claim occurred in this district.

4. This Court has personal jurisdiction over Google because Google has a place of business in, and provides infringing products and services in, the Northern District of California.

INTRADISTRICT ASSIGNMENT

5. Pursuant to Civil LR 3-2(c), this case should be subject to district-wide assignment because it is an Intellectual Property Action.

I. STATEMENT OF FACTS

A. The Plaintiff MasterObjects and its Instant Search Technology

6. From the earliest days of Internet search, the search process has been hampered by what is known as the "request-response loop." The user would type a query into a static input field, click a "submit" or "search" button, wait for the query to be sent to a remote database, wait for the result set to be returned to the server, wait for the server to build an HTML page, wait for the page to load into the browser, and then wait for the client window to be redrawn so that the result set could be viewed.

7. Inherent in the "request-response loop" is the pragmatic reality that, if the result set did not match user expectations, the entire process had to be repeated, recursively, until the results satisfied the user.

8. In 2000, Mark Smit, the founder of Plaintiff MasterObjects, invented a novel approach to search, an approach that solved the "request-response loop" problem. Smit envisioned a system where a dynamic and intelligent search field would immediately begin submitting a search query as soon as the user began typing characters into the query field. Using asynchronous communications technology, as the user typed more characters, the results in the drop-down box would change dynamically, becoming increasingly relevant as the string of characters lengthened. In essence, search would become effective and granular at the *character* level, not the block request *submit* level. More, this would happen real-time, as the user typed in characters, and not be dependent on hitting a "search" or "submit button."

9. MasterObjects' U.S. Patent No. 8,539,024 (the "'024 Patent"), entitled "System and Method for Asynchronous Client Server Session Communication," issued on September 17, 2013. Under the claims, a client object sends query messages to the server

system, with the term "query messages" being explicitly defined by the language of the claims themselves as a lengthening string of characters. *See* Claim 1, '024 Patent ("a server system, including one or more computers, which is configured to receive query messages from a client object . . . whereby the query messages represent the lengthening string . . . .). A true and correct copy of the '024 patent is attached hereto as Exhibit A. MasterObjects makes and sells products that practice the '024 patent, and MasterObjects has been selling these products from approximately 2004 forward. MasterObjects remains a going concern today, selling products that practice its patented technology.

B. <u>The Infringing Google Products</u>.

10. Google products infringe the claims of MasterObjects' '024 patent as set out below.

<u>Google Instant</u>

11. On September 8, 2010, Google launched Google "Instant." Google introduced Google Instant "as a new search enhancement that shows results as you type." Unlike the prior technology, where "you had to type a full search term, hit return, and hope for the right result," Google Instant uses asynchronous communication technology to begin sending results to the user as the user types, character-by-character. Google describes the benefit of Google Instant as follows:

> The most obvious change is that you get to the right content much faster than before because you don't have to finish typing your full search term, or even press "search." Another shift is that seeing results as you type helps you formulate a better search term by providing instant feedback. You can now adapt your search on the fly until the results match exactly what you want. In time, we may wonder how search ever worked in any other way.

Google: About Google Instant, http://www.google.com/instant.

12. In this fashion, Google Instant provides search results to users as the users type the queries. Search results are changed based on the additional characters inputted by the user, that is, as the query character string lengthens.

13. Google executives described Google Instant as representing "a fundamental shift in search," and otherwise recognized the innovative features of Google Instant in its release in September 2010.

Google Suggest

14. Google Suggest anticipates a user's query as the user types in individual characters in the query box, and asynchronously suggests complete queries that match the partial query being typed. As the user starts typing in the search box, the client asynchronously communicates with the server, and the server surveys records of previous searches to suggest potentially matching queries to the user.

15. Google describes its Google Suggest functionality as follows:

> As you type, Google's algorithm predicts and displays search queries based on other users' search activities. These searches are algorithmically determined based on a number of purely objective factors (including popularity of search terms) without human intervention. All of the predicted queries shown have been typed previously by Google users.

Google Web Search: Features: Autocomplete.

16. The benefits provided by Google Suggest parallel those provided by Google Instant, *e.g.*, speeding the search process, lessening user typing, catching mistakes mid-query, and otherwise increasing user efficiency.

Google Client Access Points for Search

17. Google makes, sells and distributes numerous client applications and platforms to provide access to its search products, including search suggestions. These

include the Chrome web browser, the Chrome operating system, the Android operating system, the Google Toolbar web browser application for Internet Explorer and Firefox, and Google Search applications for the iOS and Windows Phone platforms. Each of these client applications and platforms forms part of systems and methods that infringe the '024 Patent by, for example, returning increasingly relevant search suggestions in response to lengthening query strings input by a user.

Quick Search Box For Google Android

18. In October 2009, Google released an instant search functionality for its Android mobile phone platform, known as the "Quick Search Box."

19. As Google describes the function benefits of its quick search box:

> Since keystrokes are at a premium when you're typing on your phone, Quick Search Box provides suggestions as you type, making it easy to access whatever you're looking for by typing just a few characters. Rather than giving you one search box for the web and another for your phone, QSB provides one single search box to let you search content on your phone, including apps, contacts, and browser history, as well as content from the web, like personalized search suggestions, local business listings, stock quotes, weather, and flight status, all without opening the browser.

http://googlemobile.blogspot.com/2009/10/quick-search-box-for-android-search.html

Google Knew Of The MasterObjects Patent

20. In June of 2008, MasterObjects patent counsel sent to Marissa Mayer, a Google Vice-President responsible for Google search products, and Kent Walker, Google's General Counsel, a letter introducing Google to MasterObjects. A full and complete copy of this letter is attached as Exhibit B. The letter outlined MasterObjects' business, described MasterObjects' technology, referenced MasterObjects' website, and, included as attachments, then pending MasterObjects' applications. The letter closed by referencing a

potential merger or acquisition opportunity, or, failing that, a license.

21. Google forwarded this notice letter to senior in-house patent counsel, Laura Majerus. In 2008, Ms. Majerus was one of an intimate group of in-house Google patent prosecution lawyers. She received responsibility for Google "personalized search" patent applications in 2008. In this capacity, she worked with two other Google in-house patent lawyers, Tim Pham and Ben Lee. Mr. Pham and Mr. Lee also had responsibility for search patent applications at Google in 2008. All three were on the same e-mail alias, and all three received e-mails sent to this group alias.

22. Ms. Majerus reviewed MasterObjects' letter, clicked through to the MasterObjects website and reviewed its contents. She also reviewed the attached patent applications. She then forwarded the letter to in-house business development executive Mary Himinkool. The transmittal email summarized work that Google should do in connection with evaluating the MasterObjects purchase opportunity.

23. Shortly after receiving the email, Ms. Himinkool forwarded the material to senior Google business development executive Mike Pearson, and copied Ms. Majerus on this response.

24. In 2008, Ms. Majerus became the in-house Google patent lawyer responsible for personal search patent prosecutions. Google Suggest, and ultimately Google Instant, were search applications, and Ms. Majerus was the Google lawyer responsible for the prosecution of related Google search patent applications for at least part of calendar year 2008 through at least 2010.

25. In January 2011, Elspeth White joined Google as an in-house patent lawyer. She inherited responsibility for several existing Google search patent applications, and authorized the filing of numerous additional Google search applications, including three

explicitly relating to Google Instant, as set out below.

26. As the responsible in-house patent lawyers, Ms. Majerus and Ms. White supervised outside Google patent prosecution on the personalized search patents, including Gary S. Williams of the Morgan Lewis & Bockius firm and Paul E. Franz of the Fish & Richardson firm.

27. In 2004, Google filed an application for a predictive search technology, known as "Anticipated Query Generation and Processing in a Search Engine," now issued as U.S. Patent No. 7,836,044. Google employee Sep Kamvar was the lead inventor. Mr. Williams filed and prosecuted this application.

28. On August 28, 2008, just weeks after Ms. Majerus reviewed the MasterObjects applications and materials, Mr. Williams amended the claims in the then long outstanding and static Kamvar application. With this amendment, Google emphasized "instant search," *i.e.,* retrieving instant search results on the basis of partial and predicted queries. This is exactly the technology covered by the MasterObjects letter and applications, reviewed by Google counsel mere weeks earlier. Prior to that time, the claims in the Kamvar patent had remained largely unchanged in substance for four years.

29. By late February 2012, Google had filed four additional Kamvar applications, all covering aspects of "instant search." In these prosecutions, Google filed numerous Information Disclosure Statements. Prior to April 2012, Google did not disclose to the PTO the existence of the prior MasterObjects references.

30. Google did not forward to its outside personal search prosecution counsel, Gary S. Williams, the June 2008 MasterObjects letter. Mr. Williams saw that letter for the first time when MasterObjects' counsel marked it as an exhibit at Mr. Williams' deposition in January 2013. Nor did Google counsel inform Mr. Williams about the MasterObjects

references prior to April 2012.

31. Mr. Williams, in fact, first learned of the MasterObjects references on April 23, 2012, when Google in-house patent lawyer Elspeth White told him about the by then three MasterObjects patents, and asked Williams to file an IDS in a continuing examination on the very next day on an "urgent" basis. Ms. White knew about the MasterObjects patents and patent litigation, and knew the litigation related to instant search, no later than early April 2012. Mr. Williams testified that he would have disclosed the MasterObjects references long earlier had he been aware of those references. After being told of the references, a lawyer in Mr. Williams' firm, David Sanker, reviewed the patents, concluded that they were material, and outside counsel disclosed the references in a series of IDS's filed in pending Kamvar family applications on April 30, 2012 (but only the Kamvar family).

32. In August 2010, Google filed a new provisional captioned "Predictive Query Completion and Predictive Search Results." The first named inventor was Othar Hansson, who was a principal engineer on the Google Instant project. This application followed from work Hansson and others had done on Google Instant.

33. In August 2011, Google filed three new patent applications depending from the 2010 "Predictive Query" Google Instant provisional.

34. From the date of the August 2011 filing through the end of calendar year 2012, Google filed numerous Information Disclosure Statements in the three Google Instant applications. None of these Information Disclosure Statements disclosed the MasterObjects patents, although the Google lawyer supervising the prosecutions, Ms. Elspeth White, was fully aware of the references and indeed the underlying MasterObjects instant search patent litigation, no later than early April 2012.

35. Google outside counsel conducted an applicant initiated interview summary

with the PTO examiner on November 6, 2012 in the '134 and '135 applications, two of the Google Instant applications. In this twin application interview, as set forth in the PTO interview summary, Google counsel emphasized that none of the disclosed prior art revealed the novel elements of the claim. The MasterObjects references, which Google had carefully not disclosed to the Office, however explicitly described and anticipated the claimed Google technology.

36. The PTO issued Notices of Allowances in these two of the Google Instant applications in December 2012, calling out the points of novelty communicated by Google counsel in prior Office responses and the examiner interview. The allowance date was set as of March 25, 2013.

37. After considerable debate, Google agreed to produce Ms. Elspeth White for deposition on Tuesday, March 19, 2013. On the Thursday and Friday of the week prior, March 14 and March 15 respectively, Google outside counsel Paul E. Franz filed a Request for Continuing Examination and a new IDS form in two of the three Google Instant applications, specifically, the two allowed in December. For the first time, this post-allowance IDS disclosed the MasterObjects references.

38. In prosecuting its five Kamvar applications, several of which have now issued as patents, and the three Hansson Google Instant applications, Google represented to the PTO that its instant search claims were novel and patentable over all prior art, including the central piece of prior art Google now asserts as invalidating against the long prior but substantially identical MasterObjects patents.

39. At no time did in-house Google patent counsel responsible for the Kamvar and Hansson prosecution assess or consider whether Google Instant infringed the long-prior

MasterObjects patents. Instead, they willfully turned a blind eye to that possibility, and instead vigorously pursued competitive Google claims.

COUNT I

PATENT INFRINGEMENT (The '024 Patent)

40. MasterObjects incorporates and re-alleges, as though fully set forth herein, the allegations contained in paragraphs 1-39 above.

41. On September 17, 2013, United States Patent No. 8,539,024, entitled "System and Method for Asynchronous Client Server Session Communication," was duly and legally issued. A true and correct copy of the '024 patent is attached hereto as Exhibit A.

42. Mark Smit and Stefan van den Oord are the inventors of the '024 instant search patent. The '024 Patent has been assigned to Plaintiff. Plaintiff MasterObjects is the sole legal and rightful owner of the '024 Patent.

43. Google makes, uses, and sells products that infringe the '024 Patent, including the products described in Paragraphs 10-19 above. This conduct constitutes infringement under 35 U.S.C. § 271(a).

44. Prior even to the filing of the application that led to the '024 Patent, Google at minimum understood that there was a high probability that the MasterObjects search technology was patented and took deliberate steps to avoid knowing these and related facts, including ignoring repeated notices of pending applications related to the application that led to the '024 Patent, all in willful blindness to MasterObjects' patent portfolio and the infringing nature of Google's products with regards to that portfolio. This notice included the June 2008 letter, Exhibit B, as well as a related email sent to the then Google CEO, Eric Schmidt, in September 2008. A true and correct copy of this email is attached as Exhibit C.

Google continues to make, use and sell products that infringe the '024 Patent. Defendant's infringement of the '024 Patent is willful.

45. As a result of the infringement by Google, Plaintiff has been damaged, and will continue to be damaged, until this Defendant is enjoined from further acts of infringement.

46. Google will continue to infringe unless enjoined by this Court. Plaintiff faces real, substantial and irreparable damage and injury of a continuing nature from infringement for which Plaintiff has no adequate remedy at law.

PRAYER FOR RELIEF

WHEREFORE, Plaintiff prays for entry of judgment:

A. that the '024 Patent is valid and enforceable;

B. that Defendant has infringed one or more claims of the '024 Patent;

C. that Defendant's infringement of the claims of the '024 Patent was willful;

D. that Defendant account for and pay to Plaintiff all damages caused by the infringement of the '024 Patent, which by statute can be no less than a reasonable royalty;

E. that this Court adjudicate Defendant's infringement of the claims of the '024 Patent as willful, that the damages to Plaintiff be increased by three times the amount found or assessed pursuant to 35 U.S.C. § 284, and that the Defendant account for and pay to Plaintiff the increased amount;

F. that this Court issue a preliminary and final injunction enjoining Google, its officers, agents, servants, employees and attorneys, and any other person in active concert or participation with them, from continuing the acts herein complained of, and more particularly, that Google and such other persons be permanently enjoined and restrained from further infringing the '024 Patent;

G. that Plaintiff be granted pre-judgment and post-judgment interest on the damages caused to them by reason of Defendant's infringement of the '024 Patent;

H. that this Court require Defendant to file with this Court, within thirty (30) days after entry of final judgment, a written statement under oath setting forth in detail the manner in which Defendant has complied with the injunction;

I. that this be adjudged an exceptional case and the Plaintiff be awarded its attorney's fees in this action pursuant to 35 U.S.C. § 285;

J. that this Court award Plaintiff its costs and disbursements in this civil action, including reasonable attorney's fees; and

K. that Plaintiff be granted such other and further relief as the Court may deem just and proper under the current circumstances.

Dated: September 17, 2013 Respectfully submitted,

[illegible]

SPENCER HOSIE (CA Bar No. 101777)
shosie@hosielaw.com
DIANE S. RICE (CA Bar No. 118303)
drice@hosielaw.com
DARRELL R. ATKINSON (CA Bar No. 280564)
datkinson@hosielaw.com
HOSIE RICE LLP
Transamerica Pyramid
600 Montgomery Street, 34th Floor
San Francisco, CA 94111
(415) 247-6000 Tel.
(415) 247-6001 Fax

Attorneys for Plaintiff
MasterObjects, Inc.

DEMAND FOR JURY TRIAL

Plaintiff, by its undersigned attorneys, demands a trial by jury on all issues so triable.

Dated: September 17, 2013

Respectfully submitted,

[illegible]

SPENCER HOSIE (CA Bar No. 101777)
shosie@hosielaw.com
DIANE S. RICE (CA Bar No. 118303)
drice@hosielaw.com
DARRELL R. ATKINSON (CA Bar No. 280564)
datkinson@hosielaw.com
HOSIE RICE LLP
Transamerica Pyramid
600 Montgomery Street, 34th Floor
San Francisco, CA 94111
(415) 247-6000 Tel.
(415) 247-6001 Fax

Attorneys for Plaintiff
MasterObjects, Inc.

EXHIBIT A

US008539024B2

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

(10) **Patent No.: US 8,539,024 B2**
(45) **Date of Patent: *Sep. 17, 2013**

(54) **SYSTEM AND METHOD FOR ASYNCHRONOUS CLIENT SERVER SESSION COMMUNICATION**

(75) Inventors: **Mark H. Smit**, Maarssen (NL); **Stefan M. van den Oord**, Best (NL)

(73) Assignee: **MasterObjects, Inc.** (NL)

(*) 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.: **13/366,905**

(22) Filed: **Feb. 6, 2012**

(65) **Prior Publication Data**

US 2012/0284329 A1 Nov. 8, 2012

Related U.S. Application Data

(63) Continuation of application No. 09/933,493, filed on Aug. 20, 2001, now Pat. No. 8,112,529.

(51) **Int. Cl.**
G06F 15/16 (2006.01)

(52) **U.S. Cl.**
USPC **709/203**; 709/224; 709/227; 709/228; 709/229

(58) **Field of Classification Search**
USPC 709/203, 217, 219, 224, 227, 228, 709/229
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,255,796 A		3/1981	Gabbe et al.	
4,648,044 A		3/1987	Hardy	
4,823,310 A		4/1989	Grand	
5,444,823 A		8/1995	Nguyen	
5,572,672 A		11/1996	Dewitt et al.	
5,659,732 A		8/1997	Kirsch	
5,715,443 A		2/1998	Yanagihara	
5,724,457 A		3/1998	Fukushima	
5,748,512 A		5/1998	Vargas	
5,765,168 A		6/1998	Burrows	
5,778,381 A		7/1998	Sandifer	
5,802,292 A		9/1998	Mogul	
5,805,911 A	*	9/1998	Miller	715/234
5,845,300 A	*	12/1998	Comer et al.	715/203
5,896,321 A		4/1999	Miller	
5,978,800 A		11/1999	Yokoyama et al.	
6,006,225 A		12/1999	Bowman et al.	

(Continued)

FOREIGN PATENT DOCUMENTS

EP	1054329	11/2000
JP	8075272	5/1983
JP	H10-105562	4/1998
JP	2001-154789	6/2001

OTHER PUBLICATIONS

Andrew Clinick, Remote Scripting, Apr. 12, 1999, MSDN, pp. 1-6.*

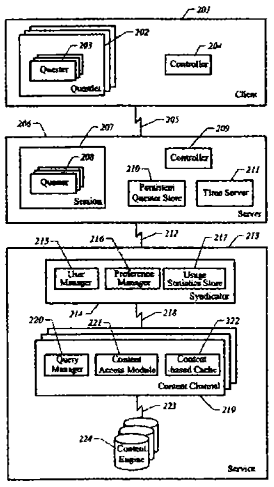
(Continued)

Primary Examiner — Barbara Burgess
(74) *Attorney, Agent, or Firm* — Fliesler Meyer LLP

(57) **ABSTRACT**

The invention provides a session-based bi-directional multi-tier client-server asynchronous information database search and retrieval system for sending a character-by-character string of data to an intelligent server that can be configured to immediately analyze the lengthening string character-by-character and return to the client increasingly appropriate database information as the client sends the string.

37 Claims, 17 Drawing Sheets



(56) **References Cited**

U.S. PATENT DOCUMENTS

6,070,184	A	5/2000	Blount	
6,078,914	A	6/2000	Redfern	
6,169,986	B1	1/2001	Bowman	
6,253,228	B1	6/2001	Ferris	
6,275,820	B1	8/2001	Navin-Chandra	
6,278,992	B1 *	8/2001	Curtis et al.	707/711
6,292,806	B1	9/2001	Sandifer	
6,347,312	B1	2/2002	Byrne	
6,356,905	B1 *	3/2002	Gershman et al.	705/26.8
6,381,593	B1	4/2002	Yano	
6,397,212	B1	5/2002	Biffar	
6,408,294	B1	6/2002	Getchius	
6,421,675	B1	7/2002	Ryan	
6,434,547	B1	8/2002	Mishelevich et al.	
6,484,162	B1	11/2002	Edlund	
6,496,833	B1	12/2002	Goldberg et al.	
6,539,379	B1 *	3/2003	Vora et al.	1/1
6,539,421	B1	3/2003	Appelman et al.	
6,564,213	B1 *	5/2003	Ortega et al.	1/1
6,578,022	B1	6/2003	Foulger	
6,629,092	B1	9/2003	Berke	
6,629,132	B1	9/2003	Ganguly	
6,633,874	B1	10/2003	Nusbickel	
6,647,383	B1	11/2003	August	
6,671,681	B1	12/2003	Emens et al.	
6,687,696	B2	2/2004	Hofmann	
6,697,849	B1	2/2004	Carlson	
6,704,727	B1	3/2004	Kravets	
6,704,906	B1	3/2004	Yankovich et al.	
6,732,090	B2	5/2004	Shanahan et al.	
6,772,150	B1	8/2004	Whitman	
6,778,979	B2	8/2004	Grefenstette et al.	
6,801,190	B1	10/2004	Robinson et al.	
6,820,075	B2	11/2004	Shanahan et al.	
6,823,514	B1	11/2004	Degenaro	
6,829,607	B1 *	12/2004	Tafoya et al.	1/1
6,832,218	B1	12/2004	Emens	
6,859,908	B1	2/2005	Clapper	
6,862,713	B1	3/2005	Kraft	
6,912,715	B2	6/2005	Gao	
6,915,279	B2	7/2005	Hogan et al.	
6,928,425	B2	8/2005	Grefenstette et al.	
6,981,215	B1	12/2005	Lindhorst	
7,000,179	B2	2/2006	Yankovich et al.	
7,039,635	B1	5/2006	Morgan	
7,043,530	B2	5/2006	Isaacs	
7,058,944	B1	6/2006	Sponheim	
7,089,228	B2	8/2006	Arnold	
7,100,116	B1	8/2006	Shafrir	
7,117,432	B1	10/2006	Shanahan et al.	
7,177,818	B2	2/2007	Nair	
7,181,459	B2	2/2007	Grant	
7,185,271	B2	2/2007	Lee	
7,216,292	B1	5/2007	Snapper	
7,240,045	B1	7/2007	Bushee	
7,251,775	B1	7/2007	Astala et al.	
7,284,191	B2	10/2007	Grefenstette et al.	
7,308,439	B2	12/2007	Baird	
7,383,299	B1 *	6/2008	Hailpern et al.	709/203
7,424,510	B2	9/2008	Gross et al.	
7,467,131	B1	12/2008	Gharachorloo	
7,499,940	B1	3/2009	Gibbs	
7,512,654	B2	3/2009	Tafoya et al.	
7,526,481	B1	4/2009	Cusson	
7,559,018	B2	7/2009	Matti	
7,610,194	B2	10/2009	Bradford	
7,647,225	B2	1/2010	Bennett et al.	
7,647,349	B2	1/2010	Hubert et al.	
7,672,932	B2	3/2010	Hood	
7,676,517	B2	3/2010	Hurst-Hiller	
7,769,757	B2	8/2010	Grefenstette et al.	
7,788,248	B2	8/2010	Forstall	
7,836,044	B2	11/2010	Kamvar et al.	
7,840,557	B1	11/2010	Smith	
7,840,589	B1	11/2010	Holt	
7,856,432	B2	12/2010	Tesch et al.	
7,890,516	B2	2/2011	Zarzar Charur et al.	
7,890,526	B1	2/2011	Brewer	
7,900,228	B2	3/2011	Stark et al.	
7,941,819	B2	5/2011	Stark et al.	
8,131,258	B2	3/2012	Smith et al.	
8,135,729	B2	3/2012	Brewer et al.	
2001/0049676	A1	12/2001	Kepler	
2002/0049756	A1 *	4/2002	Chua et al.	707/4
2002/0065879	A1	5/2002	Ambrose et al.	
2002/0069122	A1	6/2002	Yun et al.	
2002/0129012	A1 *	9/2002	Green	707/3
2002/0138571	A1	9/2002	Trinon et al.	
2002/0138640	A1	9/2002	Raz et al.	
2003/0033288	A1	2/2003	Shanahan et al.	
2003/0041058	A1	2/2003	Ibuki et al.	
2003/0061200	A1	3/2003	Hubert et al.	
2003/0071850	A1	4/2003	Geidl	
2003/0120554	A1	6/2003	Hogan et al.	
2004/0093562	A1	5/2004	Diorio et al.	
2004/0141011	A1	7/2004	Smethers et al.	
2004/0142720	A1	7/2004	Smethers	
2004/0205448	A1	10/2004	Grefenstette et al.	
2005/0022114	A1	1/2005	Shanahan et al.	
2005/0055438	A1	3/2005	Matti	
2005/0120005	A1	6/2005	Tesch et al.	
2005/0283468	A1	12/2005	Kamvar et al.	
2006/0004843	A1	1/2006	Tafoya et al.	
2006/0026636	A1	2/2006	Stark et al.	
2006/0026638	A1	2/2006	Stark et al.	
2006/0031880	A1	2/2006	Stark et al.	
2006/0041927	A1	2/2006	Stark et al.	
2006/0184546	A1 *	8/2006	Yano et al.	707/10
2007/0050351	A1	3/2007	Kasperski et al.	
2007/0050352	A1	3/2007	Kim	
2007/0143262	A1	6/2007	Kasperski	
2007/0288648	A1	12/2007	Mehanna et al.	
2008/0071561	A1	3/2008	Holcombe	
2008/0147788	A1	6/2008	Omoigui	
2010/0267362	A1	10/2010	Smith et al.	
2011/0106831	A1	5/2011	Zarzar Charur et al.	
2011/0173217	A1	7/2011	Kasperski	
2011/0320472	A1	12/2011	Griffith et al.	

OTHER PUBLICATIONS

Anonymous, Ajax (Programming), Wikipedia.org, XP-002401064, Retrieved from the Internet: <http://www.en.wikipedia.org/wiki/Ajax,sub.—(programming)>.

International Searching Authority, International Search Report for PCT/US02/25729, Nov. 5, 2002, 3 pages.

Harless, Membership Database on USA Gymnastics Online, 1996, 5 pages Retrieved from the Internet: URL: http://usa-gymnastics.org/publications/technique/1996/9/membership-query.html.

Nareddy, Introduction to Microsoft Index Server, Oct. 15, 1997, 9 pages Retrieved from the Internet: URL: http://msdn.microsoft.com/en-us/library/ms951563(printer).aspx.

Clinick, Remote Scripting, Apr. 12, 1999, Microsoft Corporation, 6 pages Retrieved from the Internet: URL: http://msdn.microsoft.com/en-us/library/ms951563(printer).aspx.

Masterobjects, Inc., Introducing QuestObjects, 2006, XP002496891, 25 pages Retrieved from the Internet: URL: http://www.questobjects.masterobjects.com/documents/go-introducing.pdf.

European Patent Office, European Search Report for European Patent Application No. EP08252534.6-1225, Oct. 14, 2008, 9 pages.

European Patent Office, European Examination Report for European Patent Application No. EP02763441.9, 4 pages.

European Patent Office, European Search Report for European Patent Application No. EP02763441.9, 3 pages.

Widjaja, Communication Networks, Fundamental Concepts and Key Architecture, 2004, pp. 315-316 and 611-612, McGraw-Hill, 2nd Ed.

Marsch, Remote Scripting, XP002401062, Retrieved from the Internet: <http://www.microsoft. com/germany/msdn/library/web/RemoteScripting.mspx?pf-=true>.

Anonymous, Using the XML HTTP Request Object, XP-002401063, Retrieved from the Internet: <http://www.jibbering.com/2002/4/httpre quest.2002.html>.

Doherty, Web-based E-Mail, May 29, 2000, 3 pages. Retrieved from: http://www.networkcomputing.com/1110/1110f3.html?Is=NCJS_1110bt.
Cheong, et al., A Boolean Query Processing with a Result Cache in Mediator Systems, Advances in Digital Libraries, May 22-24, 2000, 10 pages.
Jakobsson, Autocompletion in Full Text Transaction Entry: A Method for Humanized Input, 1986, vol. 17.
Livingston, Windows 98 Secrets, 1998, pp. 232-235.
Markatos, et al., On Caching Search Engine Results, May 2000, 23 pages.
Krishnamurthy, et al., Web Protocols and Practice : HTTP/1.1, Networking Protocols, Caching and Traffic Measurement, 2001.
Kientzle, A JAVA Applet Search Engine, Feb. 1999.
Homer, XMLin IE5 Programmers Reference, 1999.
Xia, et al.. Supporting Web-Based Database Application Development, 1999, 8 pages.
Chen, et al., The Implementation and Performance Evaluation of the ADMS Query Optimizer: Integrating Query Result Caching and Matching, Oct. 1993, 21 pages.
Unknown Author, Netscape Communicator for Solaris 4.7 Release Notes, Aug. 20, 1999, 5 pages.
Oracle International Corporation, iPlanet Directory Server 4.11 LDAP Setup and Configuration Guide, Chapter 3, 2001, 14 pages.
Netscape Communications Corporation, Netscape Directory Server 4.1 Deployment, Administrators Guide, 1999.
Kapitskaia, et al., Evolution and Revolution in LDAP Directory Caches, Advances in Database Technology—EDBT, 2000, pp. 202-216.
Glick, Global Address Book and LDAP UI Proposal, 2001.
Unknown Author, Mozilla 0.9.1 Release Notes, 2001, 23 pages.
Giovetti, Microsoft Money, COMPUTE!, Jul. 1992, p. 105, Issue 142.
Microsoft Corporation, MSN Hotmail: From Zero to 30 Million Members in 30 Months, Feb. 8, 1999.
Qualcomm, Inc., Qualcomm Extends Internet E-mail Presence to the Web, Dec. 10, 1997.
Johnson, et al., A Hypertextual Interface for a Searcher's Thesaurus, Jun. 11-13, 1995, 15 pages.
Deadmond, Address Book: What a Concept!, Jun. 1, 1999, 2 pages.
Hassan, Stanford Digital Library Interoperability Protocol, 1997, 42 pages.
Buyukkoten, Focused Web Searching with PDAs, May 15-19, 2000, 21 pages.
O'Brien, The New Domino R5 Directory Catalog: An Administrator's Guide, Nov./Dec. 1998.
Beaulie, et al., Okapi at TREC-5, Jan. 31, 1997, 23 pages.
Jones, Graphical Query Specification and Dynamic Result Previews for a Digital Library, 1998, 9 pages.
Jones, Dynamic Query Result Previews for a Digital Library, Jun. 1998, 3 pages.
Unkown Author, Using Netscape Communicator at Lehigh, 15 pages, retrieved from the World Wide Web: http://web.archive.org/web/20001002224731/http:/www.lehigh.edu/~inhelp/faq/qa/nsfiles/nsfall2000-2.htm.

* cited by examiner

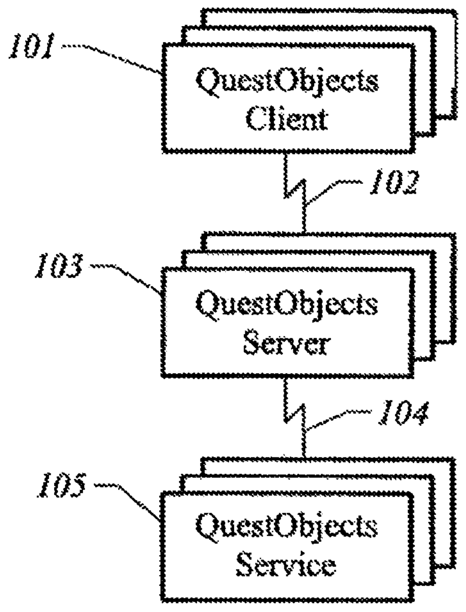

101
QuestObjects Client
102
103
QuestObjects Server
104
105
QuestObjects Service

FIG. 1

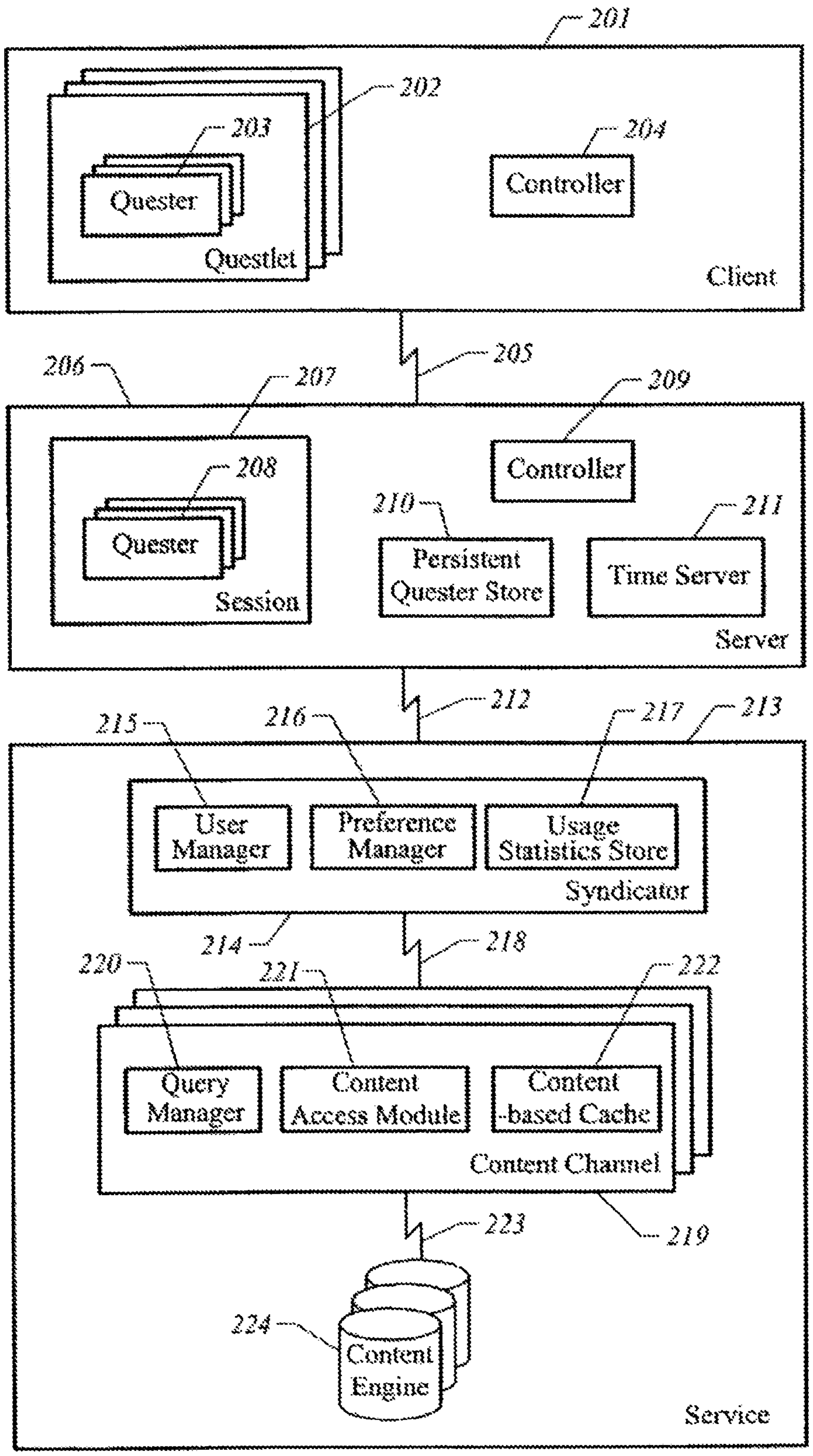

201
202
203
204
Quester
Questlet
Controller
Client
205
206
207
208
209
Controller
Quester
Session
210
Persistent Quester Store
211
Time Server
Server
212
213
215
216
217
User Manager
Preference Manager
Usage Statistics Store
Syndicator
214
218
220
221
222
Query Manager
Content Access Module
Content -based Cache
Content Channel
219
223
224
Content Engine
Service

FIG. 2

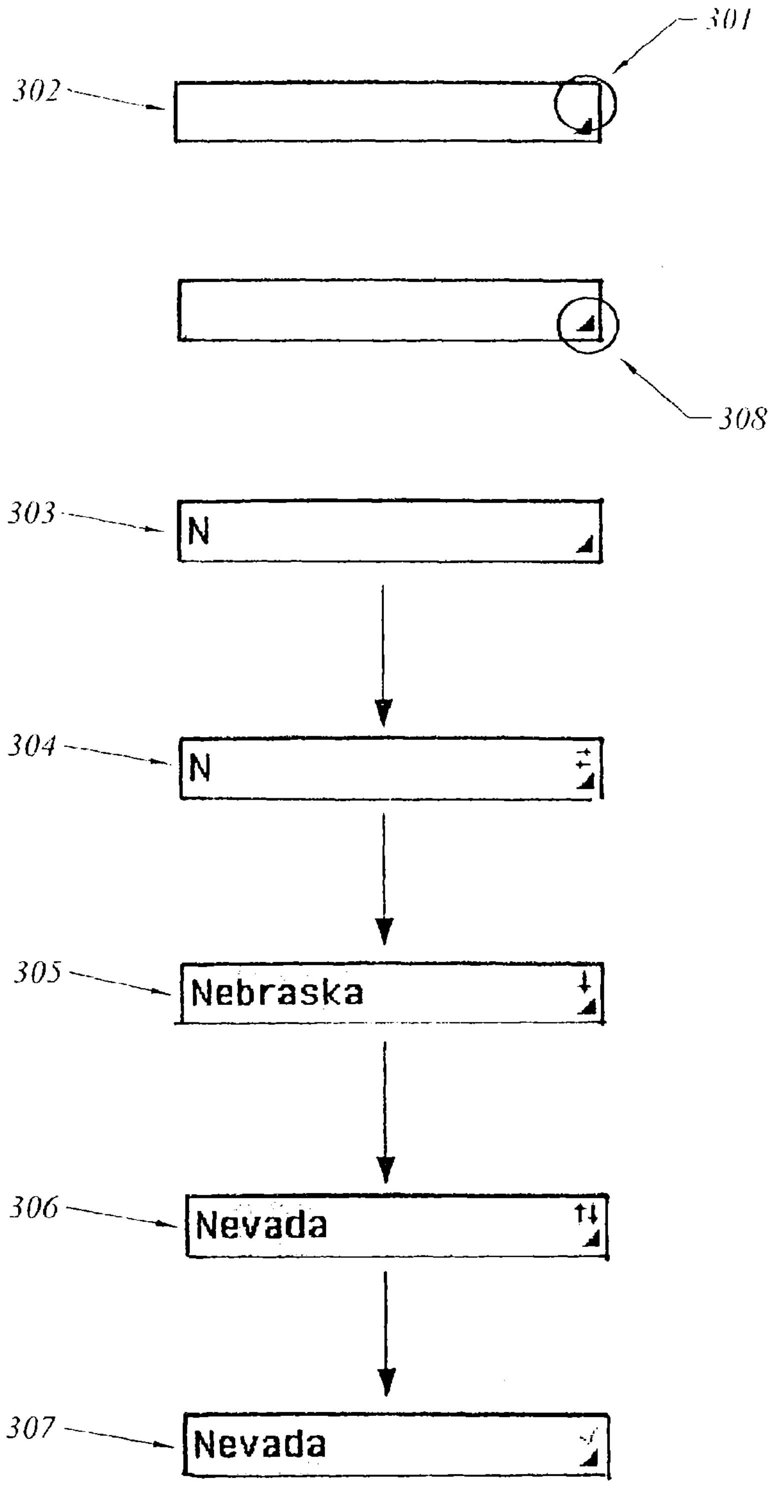


FIG. 3A

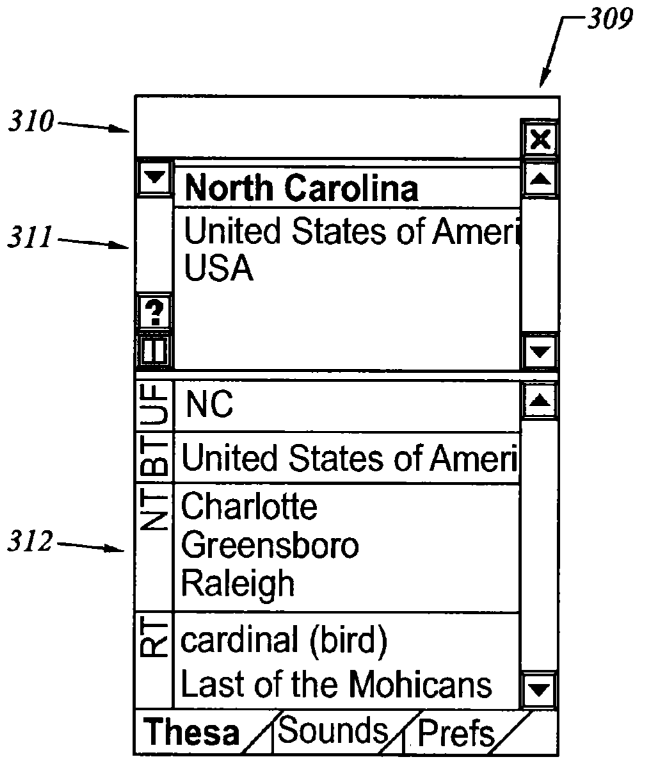

309
310
311
312
North Carolina
United States of Ameri
USA
UF NC
BT United States of Ameri
NT Charlotte
Greensboro
Raleigh
RT cardinal (bird)
Last of the Mohicans
Thesa
Sounds
Prefs

FIG. 3B

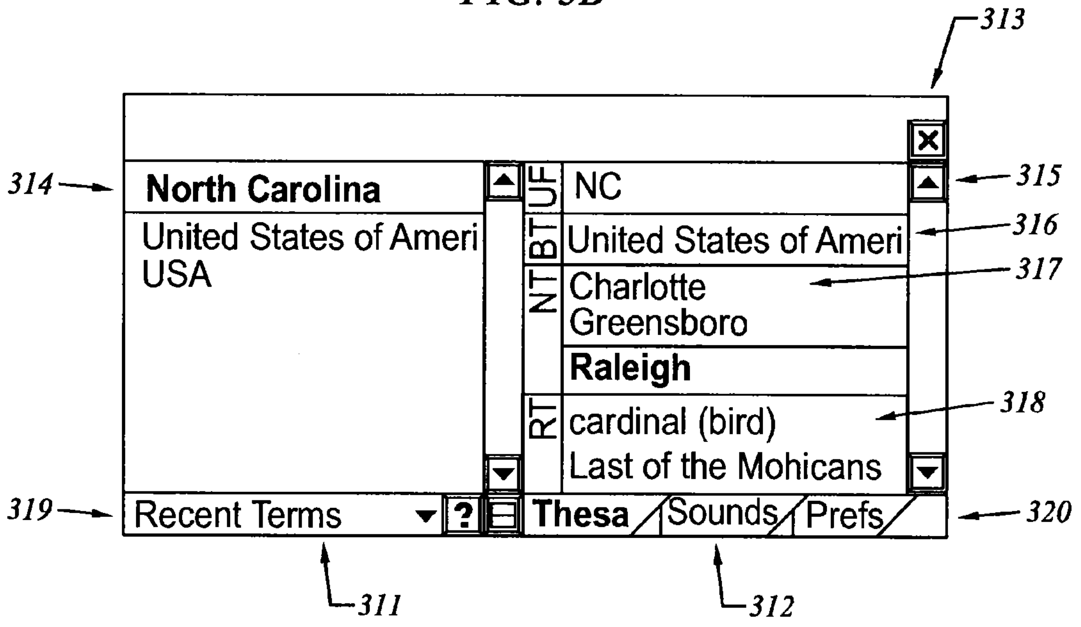

313
314
315
316
317
318
319
320
311
312
North Carolina
United States of Ameri
USA
UF NC
BT United States of Ameri
NT Charlotte
Greensboro
Raleigh
RT cardinal (bird)
Last of the Mohicans
Recent Terms
Thesa
Sounds
Prefs

FIG. 3C

U.S. Patent Sep. 17, 2013 Sheet 5 of 17 US 8,539,024 B2

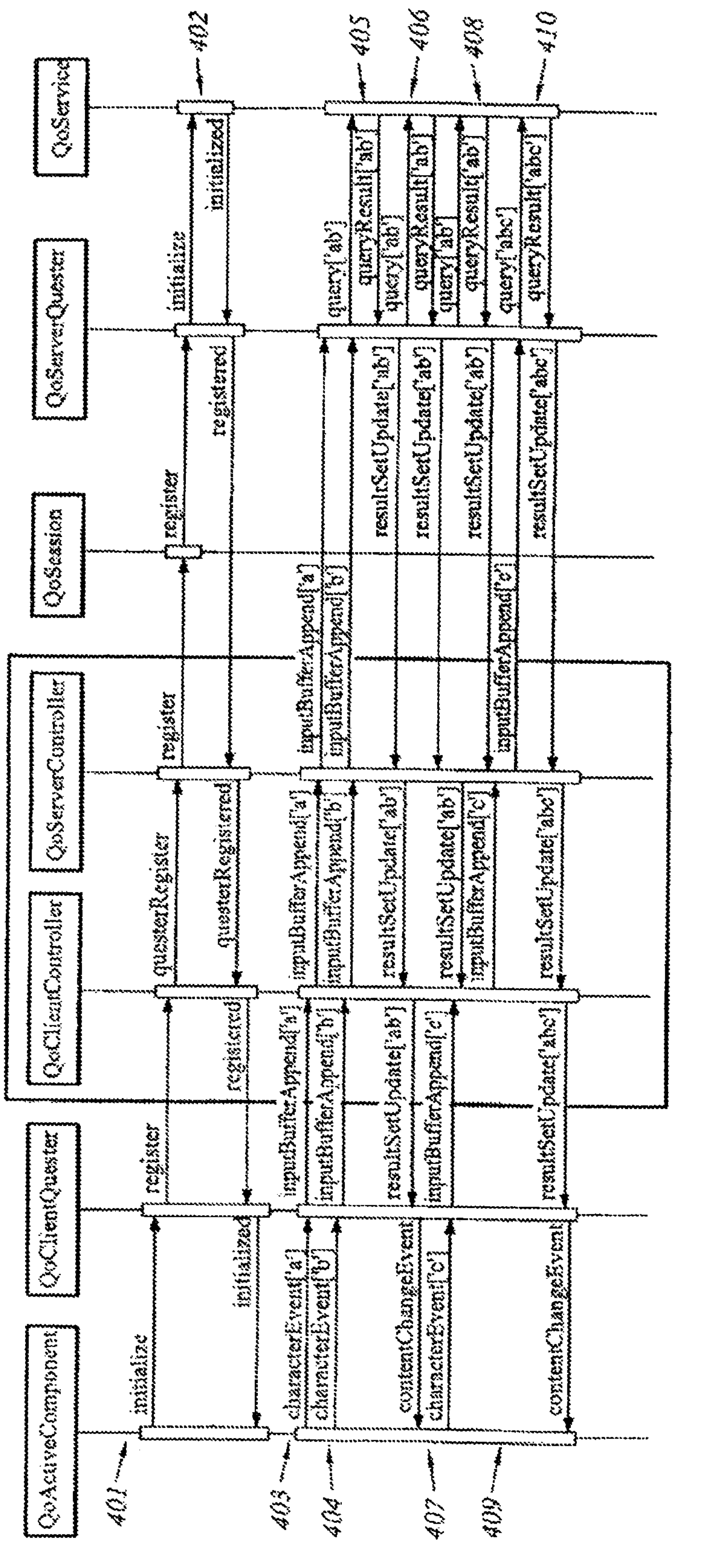


FIG. 4

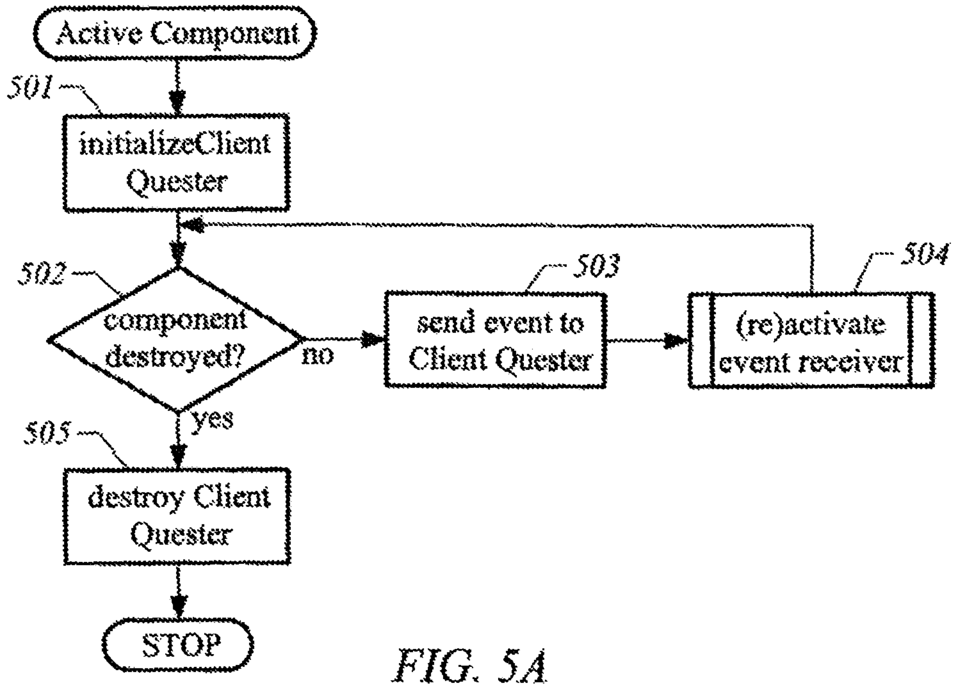

Active Component
501
initializeClient Quester
502
component destroyed?
no
503
send event to Client Quester
504
(re)activate event receiver
yes
505
destroy Client Quester
STOP

FIG. 5A

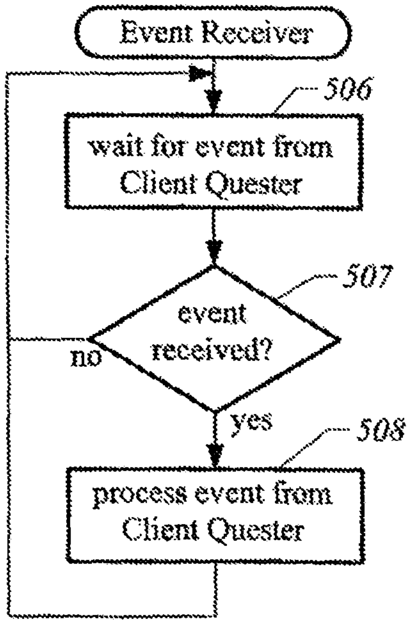

Event Receiver
506
wait for event from Client Quester
507
event received?
no
yes
508
process event from Client Quester

FIG. 5B

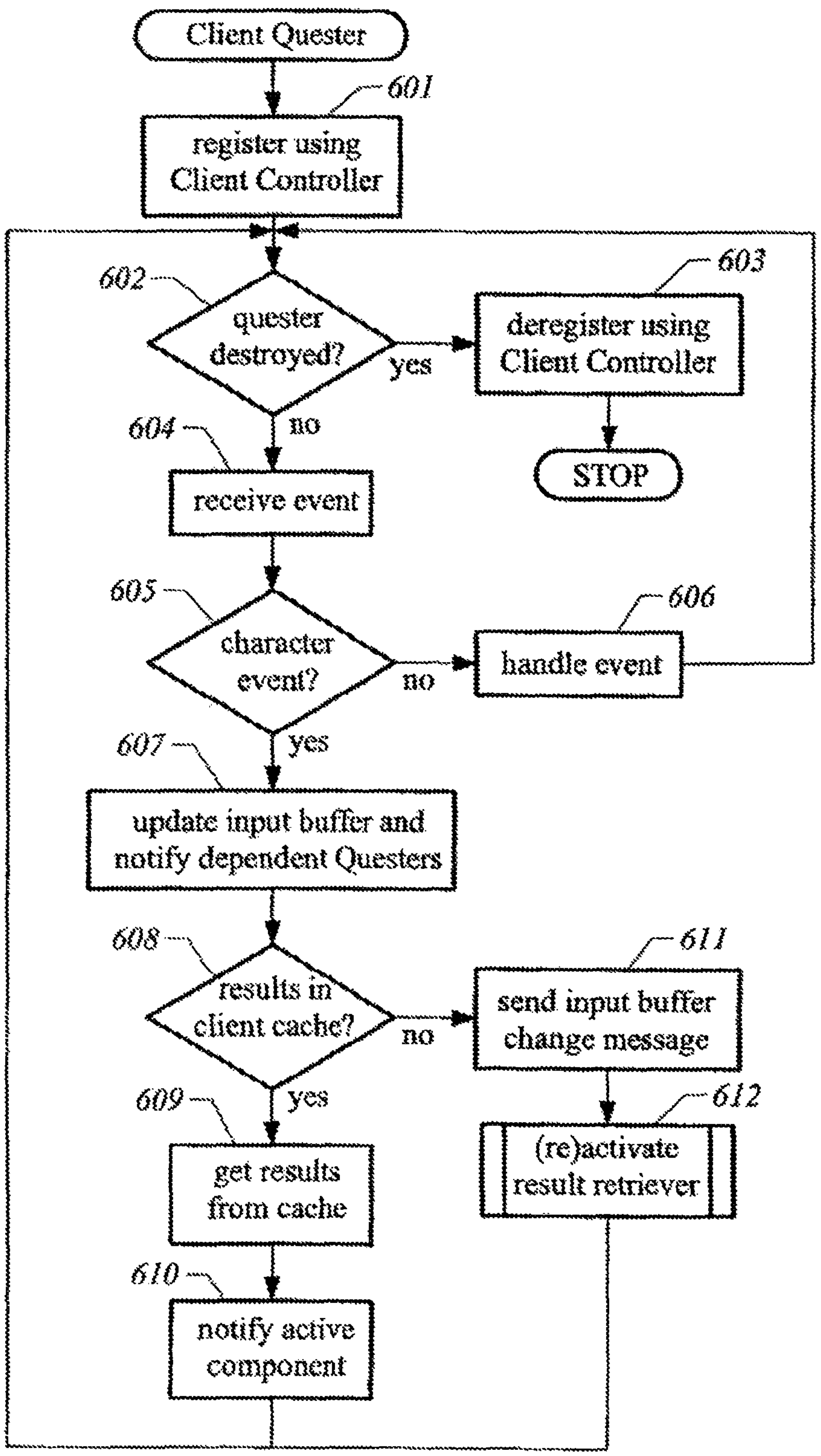

Client Quester
601
register using Client Controller
602
quester destroyed?
yes
no
603
deregister using Client Controller
STOP
604
receive event
605
character event?
no
yes
606
handle event
607
update input buffer and notify dependent Questers
608
results in client cache?
no
yes
611
send input buffer change message
612
(re)activate result retriever
609
get results from cache
610
notify active component

FIG. 6A

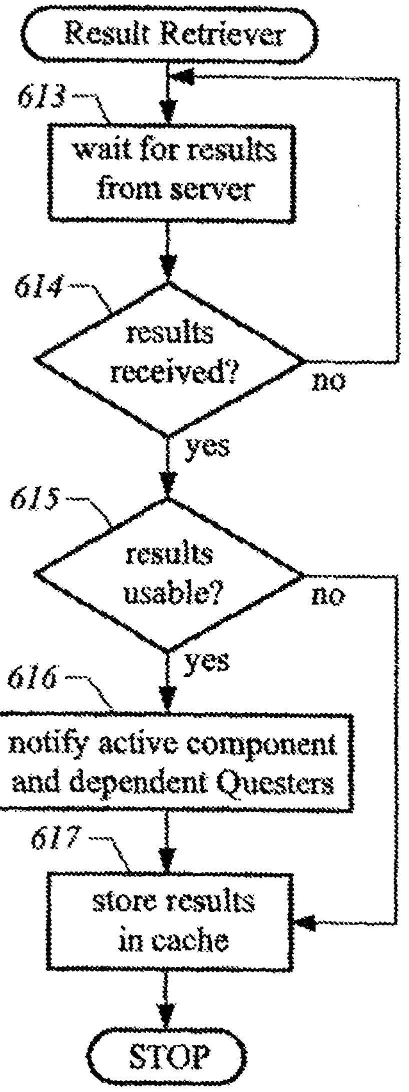

Result Retriever
613
wait for results from server
614
results received?
no
yes
615
results usable?
no
yes
616
notify active component and dependent Questers
617
store results in cache
STOP

FIG. 6B

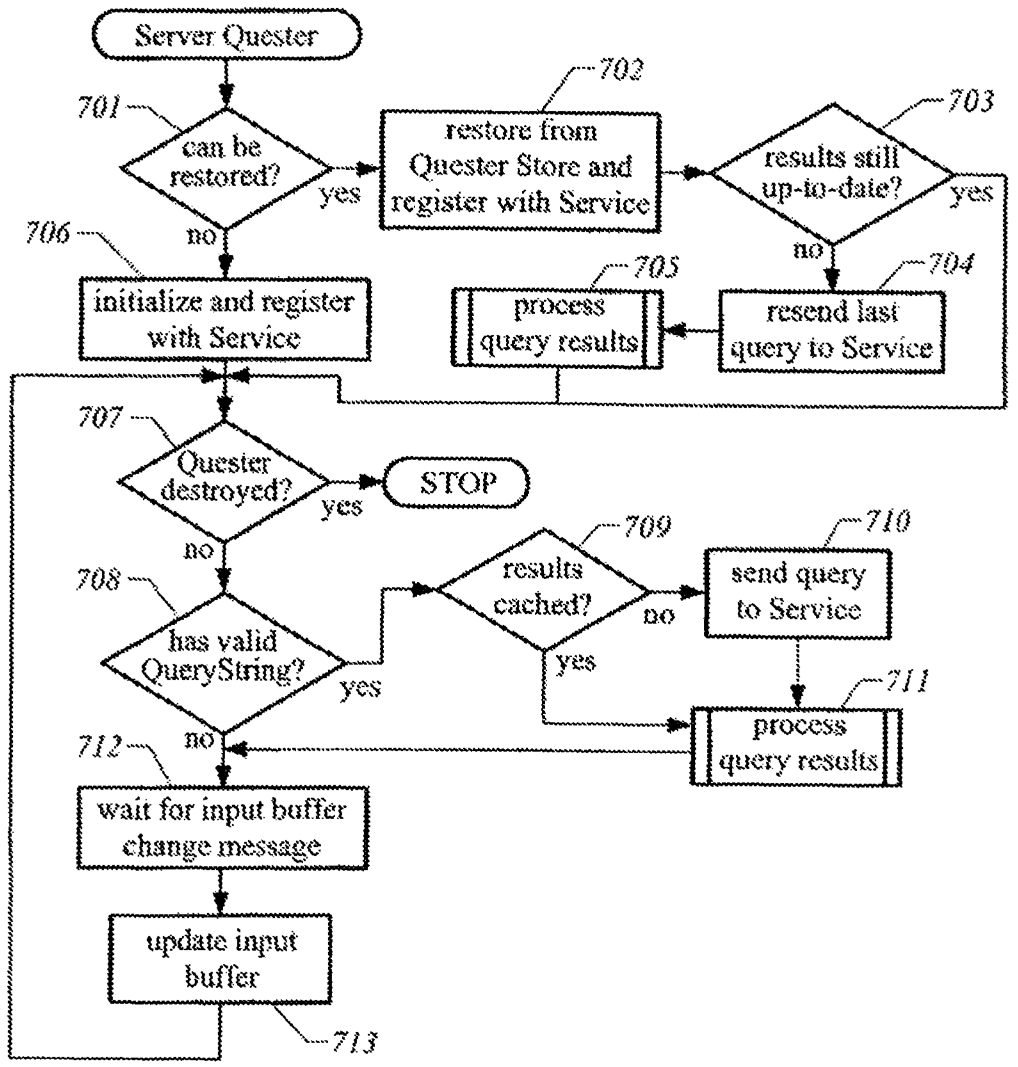

Server Quester
701
can be restored?
yes
no
702
restore from Quester Store and register with Service
703
results still up-to-date?
yes
no
706
initialize and register with Service
705
process query results
704
resend last query to Service
707
Quester destroyed?
yes
STOP
no
708
has valid QueryString?
yes
no
709
results cached?
no
yes
710
send query to Service
711
process query results
712
wait for input buffer change message
update input buffer
713

FIG. 7A

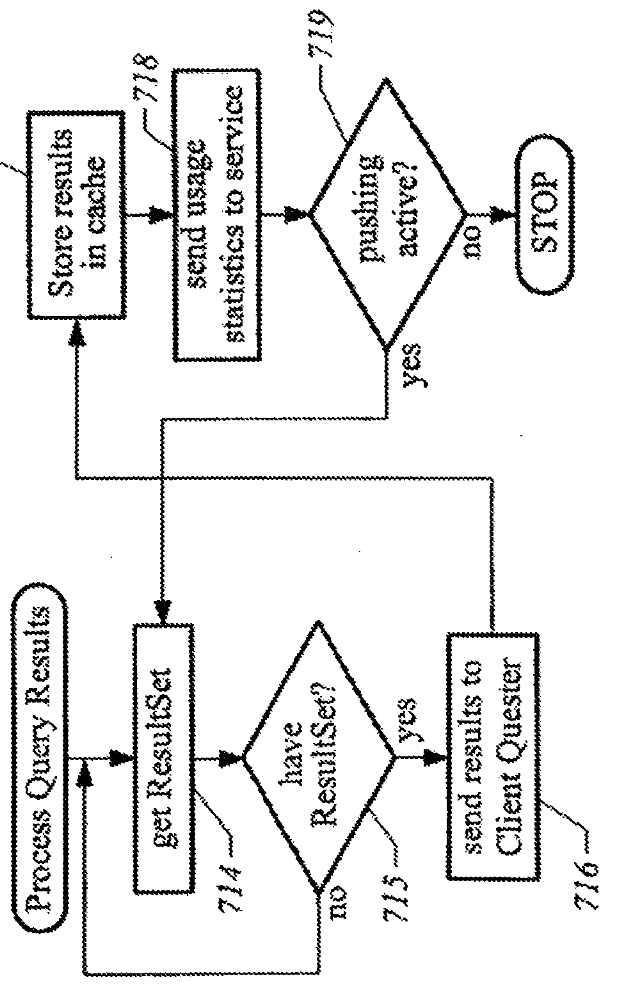

Process Query Results
get ResultSet
714
have ResultSet?
715
no
yes
send results to Client Quester
716
Store results in cache
717
send usage statistics to service
718
pushing active?
719
yes
no
STOP

FIG. 7B

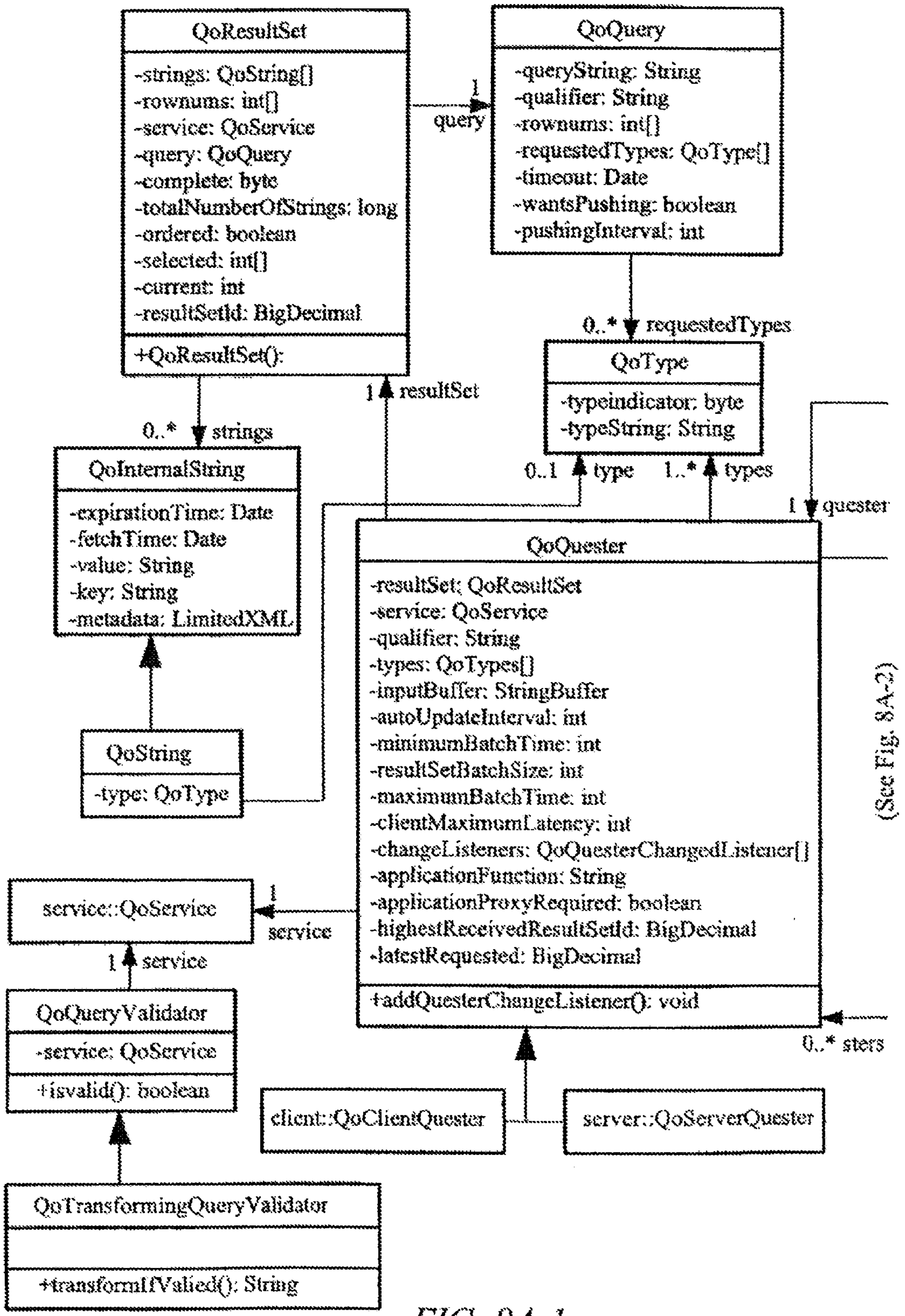

QoResultSet
-strings: QoString[]
-rownums: int[]
-service: QoService
-query: QoQuery
-complete: byte
-totalNumberOfStrings: long
-ordered: boolean
-selected: int[]
-current: int
-resultSetId: BigDecimal
+QoResultSet():
1
query
QoQuery
-queryString: String
-qualifier: String
-rownums: int[]
-requestedTypes: QoType[]
-timeout: Date
-wantsPushing: boolean
-pushingInterval: int
0..* requestedTypes
QoType
-typeindicator: byte
-typeString: String
1 resultSet
0..* strings
QoInternalString
-expirationTime: Date
-fetchTime: Date
-value: String
-key: String
-metadata: LimitedXML
0..1 type
1..* types
1 quester
QoQuester
-resultSet: QoResultSet
-service: QoService
-qualifier: String
-types: QoTypes[]
-inputBuffer: StringBuffer
-autoUpdateInterval: int
-minimumBatchTime: int
-resultSetBatchSize: int
-maximumBatchTime: int
-clientMaximumLatency: int
-changeListeners: QoQuesterChangedListener[]
-applicationFunction: String
-applicationProxyRequired: boolean
-highestReceivedResultSetId: BigDecimal
-latestRequested: BigDecimal
+addQuesterChangeListener(): void
QoString
-type: QoType
(See Fig. 8A-2)
service::QoService
1
service
1 service
QoQueryValidator
-service: QoService
+isvalid(): boolean
0..* sters
client::QoClientQuester
server::QoServerQuester
QoTransformingQueryValidator
+transformIfValied(): String

FIG. 8A-1

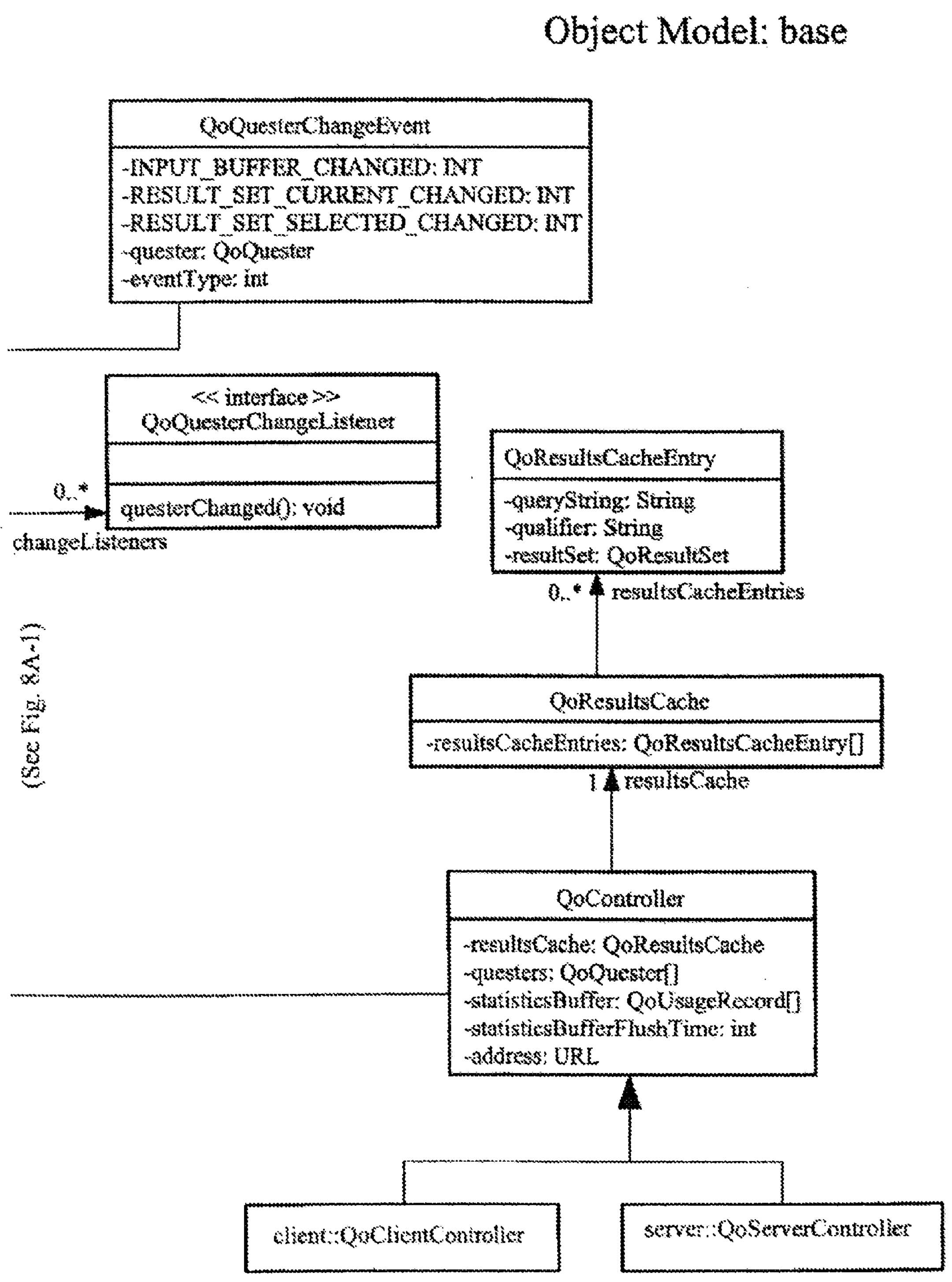

Object Model: base
QoQuesterChangeEvent
-INPUT_BUFFER_CHANGED: INT
-RESULT_SET_CURRENT_CHANGED: INT
-RESULT_SET_SELECTED_CHANGED: INT
-quester: QoQuester
-eventType: int
<< interface >>
QoQuesterChangeListener
questerChanged(): void
0..*
changeListeners
QoResultsCacheEntry
-queryString: String
-qualifier: String
-resultSet: QoResultSet
0..*
resultsCacheEntries
(See Fig. 8A-1)
QoResultsCache
-resultsCacheEntries: QoResultsCacheEntry[]
1
resultsCache
QoController
-resultsCache: QoResultsCache
-questers: QoQuester[]
-statisticsBuffer: QoUsageRecord[]
-statisticsBufferFlushTime: int
-address: URL
client::QoClientController
server::QoServerController

FIG. 8A-2

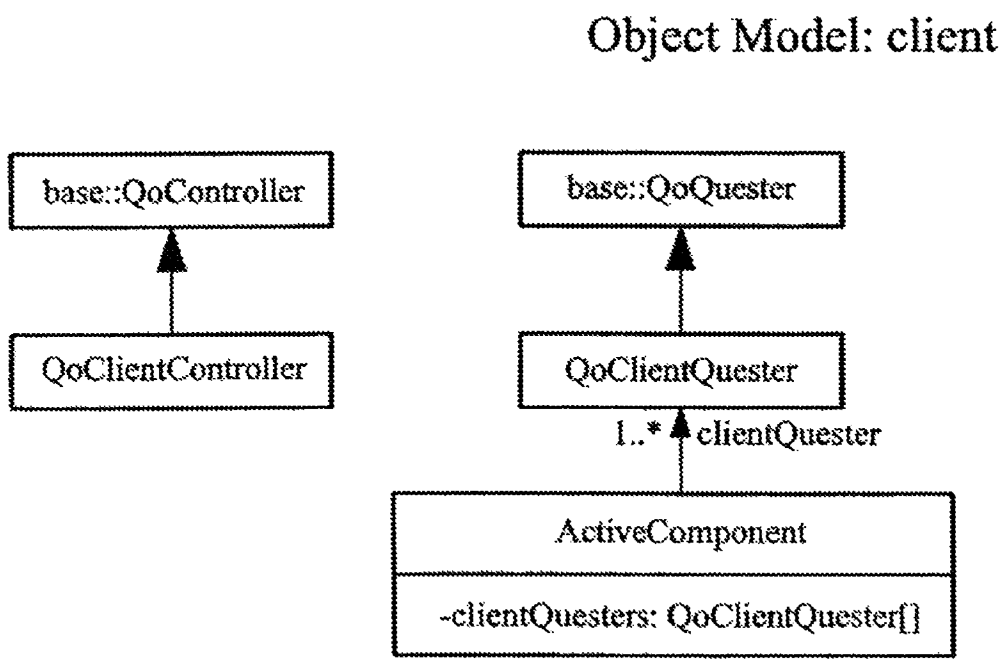

Object Model: client
base::QoController
QoClientController
base::QoQuester
QoClientQuester
1..* clientQuester
ActiveComponent
-clientQuesters: QoClientQuester[]

FIG. 8B

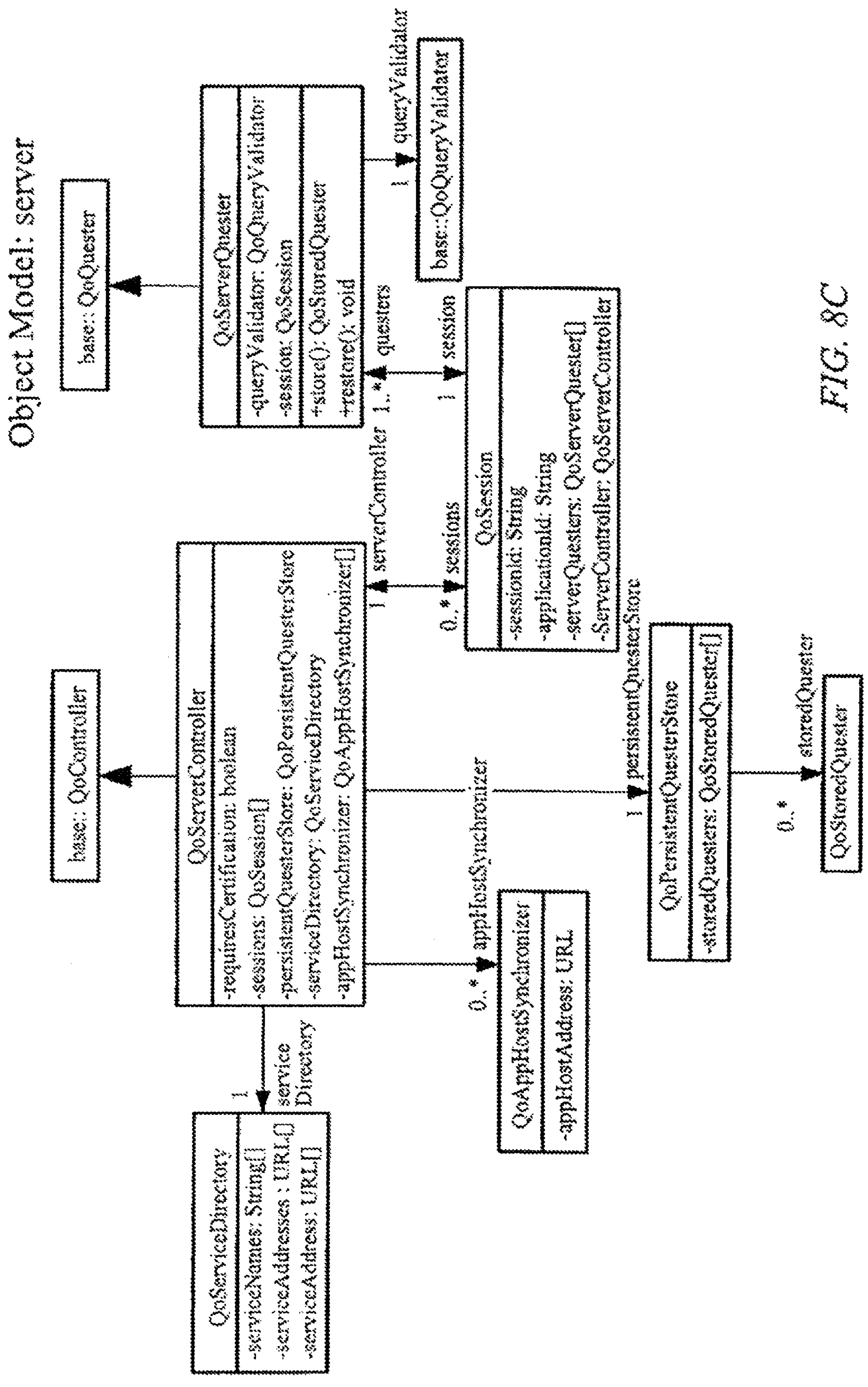

Object Model: server
base:: QoQuester
QoServerQuester
-queryValidator: QoQueryValidator
-session: QoSession
+store(): QoStoredQuester
+restore(): void
1
queryValidator
base::QoQueryValidator
1..*
questers
1
session
QoSession
-sessionId: String
-applicationId: String
-serverQuesters: QoServerQuester[]
-ServerController: QoServerController
1
serverController
0..*
sessions
base:: QoController
QoServerController
-requiresCertification: boolean
-sessions: QoSession[]
-persistentQuesterStore: QoPersistentQuesterStore
-serviceDirectory: QoServiceDirectory
-appHostSynchronizer: QoAppHostSynchronizer[]
1
persistentQuesterStore
QoPersistentQuesterStore
-storedQuesters: QoStoredQuester[]
0..*
storedQuester
QoStoredQuester
0..*
appHostSynchronizer
QoAppHostSynchronizer
-appHostAddress: URL
1
service Directory
QoServiceDirectory
-serviceNames: String[]
-serviceAddresses : URL[]
-serviceAddress: URL[]

FIG. 8C

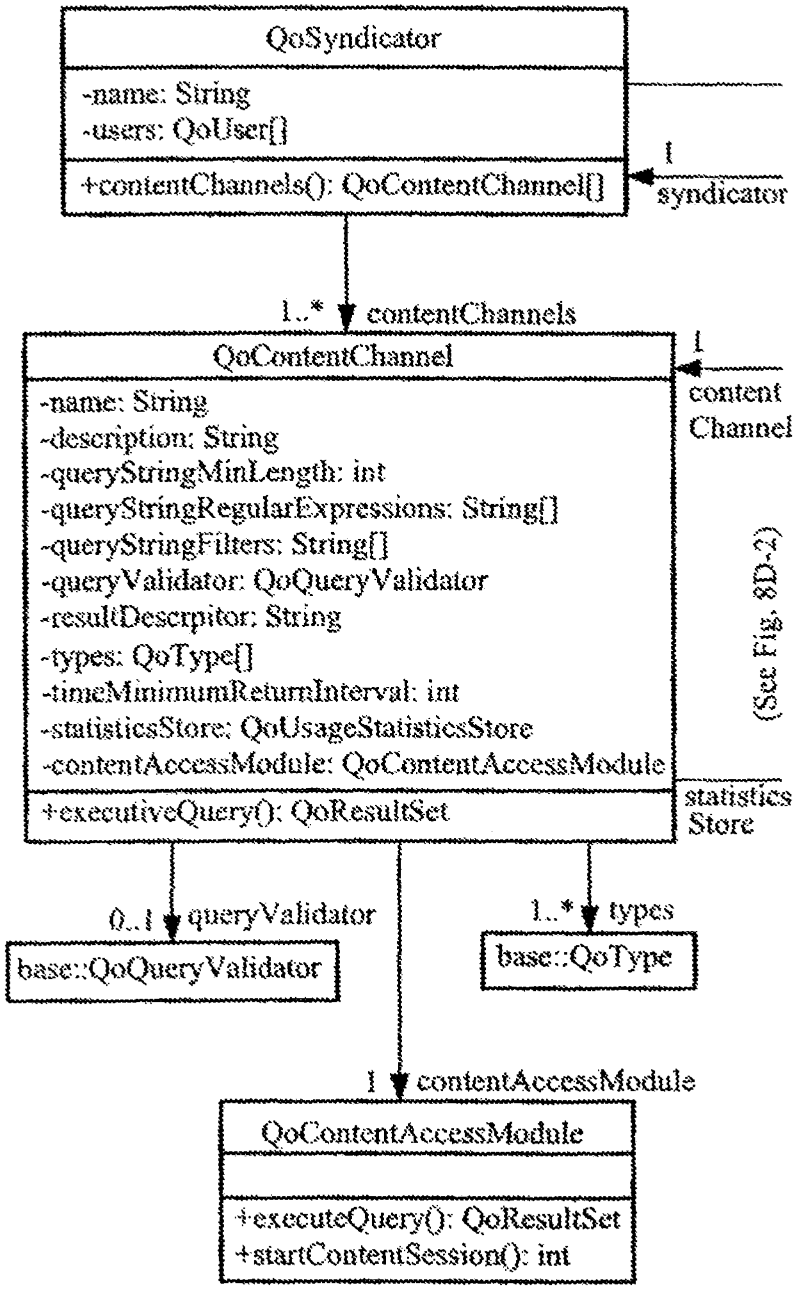

QoSyndicator
-name: String
-users: QoUser[]
+contentChannels(): QoContentChannel[]
1
syndicator
1..* contentChannels
QoContentChannel
-name: String
-description: String
-queryStringMinLength: int
-queryStringRegularExpressions: String[]
-queryStringFilters: String[]
-queryValidator: QoQueryValidator
-resultDescrpitor: String
-types: QoType[]
-timeMinimumReturnInterval: int
-statisticsStore: QoUsageStatisticsStore
-contentAccessModule: QoContentAccessModule
+executiveQuery(): QoResultSet
1
content Channel
(See Fig. 8D-2)
statistics Store
0..1 queryValidator
base::QoQueryValidator
1..* types
base::QoType
1 contentAccessModule
QoContentAccessModule
+executeQuery(): QoResultSet
+startContentSession(): int

FIG. 8D-1

Object Model: service

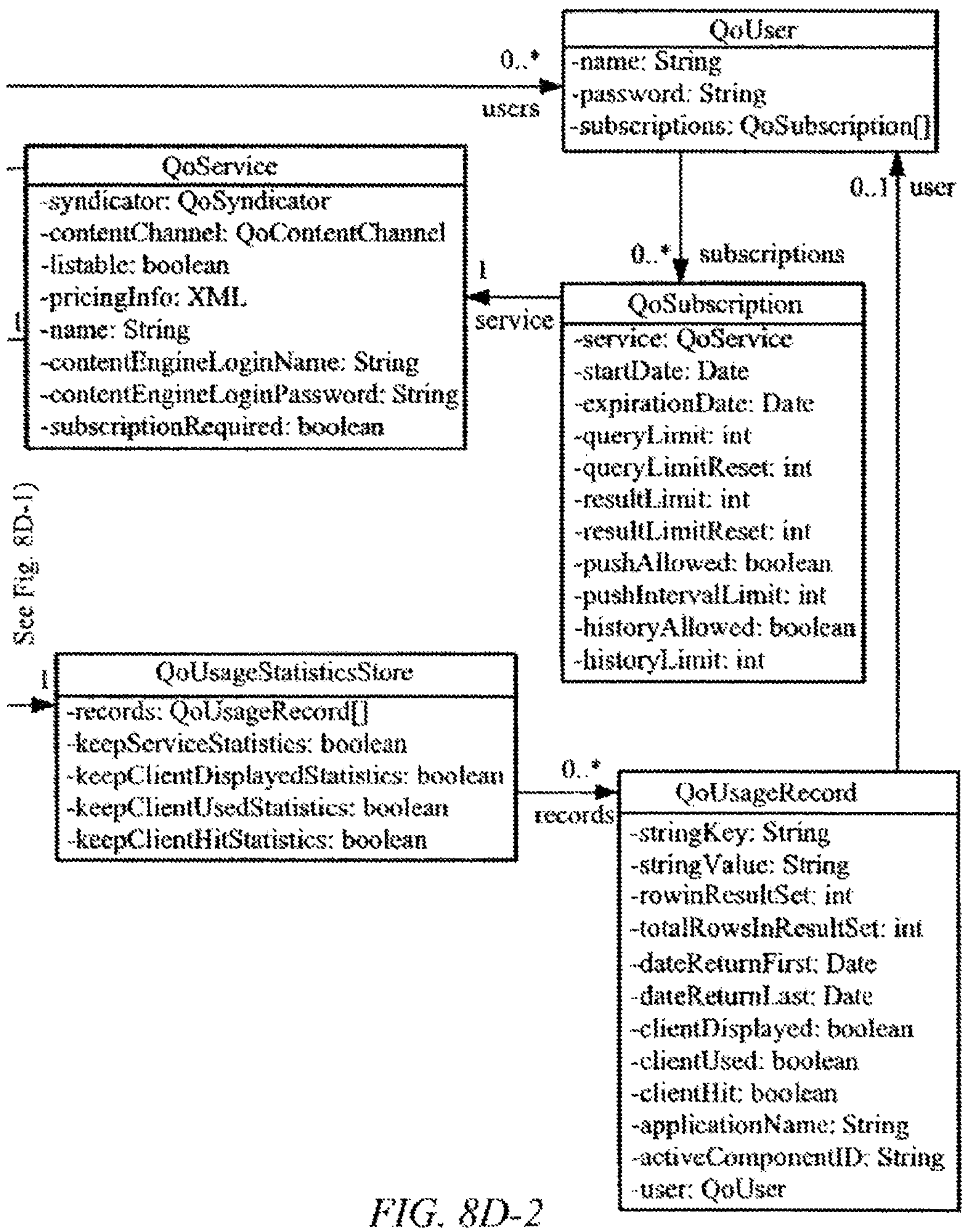


FIG. 8D-2

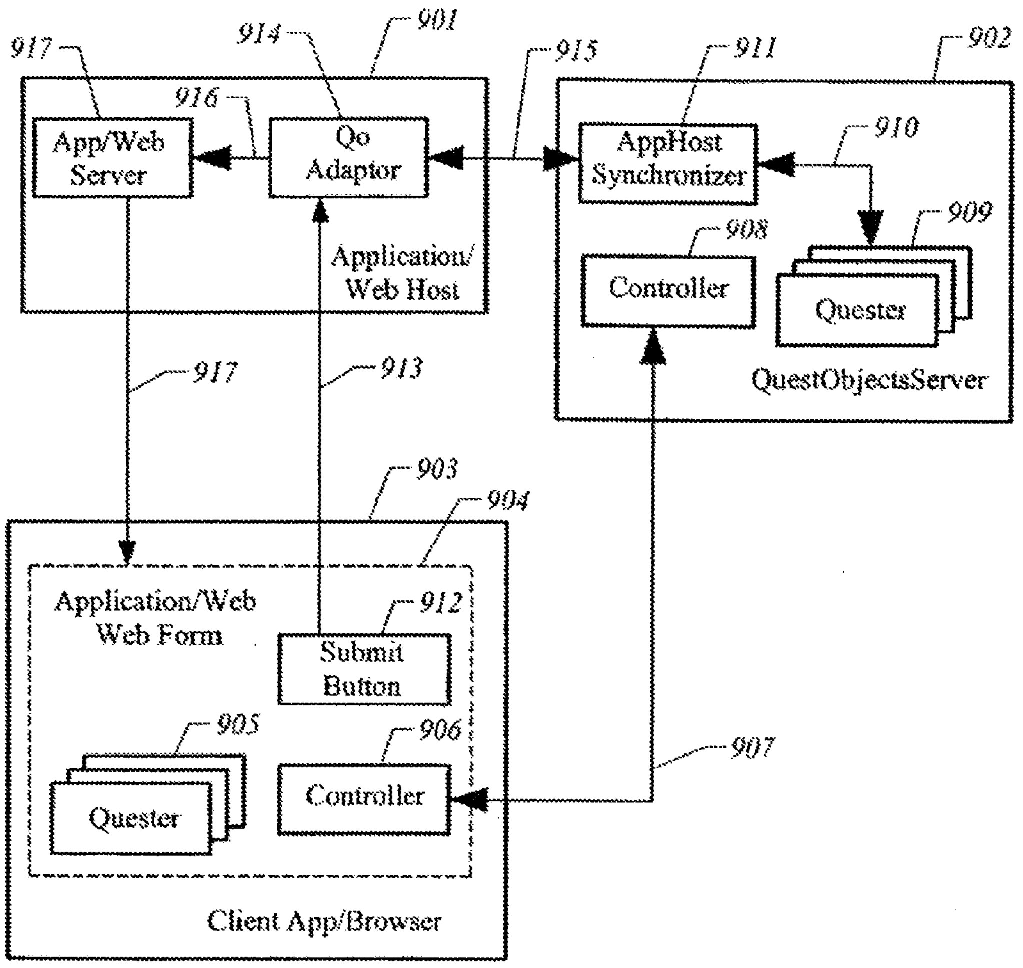

917
914
901
915
911
902
916
App/Web Server
Qo Adaptor
AppHost Synchronizer
910
909
908
Controller
Quester
Application/ Web Host
917
913
QuestObjectsServer
903
904
Application/Web Web Form
912
Submit Button
905
906
907
Quester
Controller
Client App/Browser

FIG. 9

SYSTEM AND METHOD FOR ASYNCHRONOUS CLIENT SERVER SESSION COMMUNICATION

CLAIM OF PRIORITY

This application is a continuation of U.S. patent application Ser. No. 09/933,493, filed on Aug. 20, 2001 entitled: "SYSTEM AND METHOD FOR ASYNCHRONOUS CLIENT SERVER SESSION COMMUNICATION", by Mark H. Smit, et al, now U.S. Pat. No. 8,112,529, issued on Feb. 7, 2012, which is incorporated herein by reference.

COPYRIGHT NOTICE

FIELD OF THE INVENTION

The invention relates generally to client-server communication systems, and particularly to a session-based bi-directional multi-tier client-server asynchronous search and retrieval system.

BACKGROUND OF THE INVENTION

A primary task of computer systems is to manage large quantities of information, generally referred to as data. The first computers typically stored data using off-line methods, for example by using punch cards and other primitive means. As built-in or on-line storage solutions became more affordable, data were instead stored in central memory banks. The first enterprise-wide computer systems consisted of central computers containing central data storage, and a large number of user terminals that accessed this server data by sending input and receiving output as characters to be displayed or printed at the terminal. Although these systems had a primitive user interface and data access became increasingly slower as the number of users grew, these systems nevertheless handled enterprise data with ease and great security.

The first servers, often referred to as mainframes or mini computers, ran on proprietary operating systems. Terminals usually had large input buffers where input was only checked against or committed to the server after entering text into a page or form. Many systems only displayed the character entered after it was received and confirmed by the server. Faster servers and more modern server operating systems, such as Unix and VMS, offered several advantages in that users could receive immediate feedback after each character was typed.

At the beginning of the 1980s decade, the growing popularity of microcomputers and personal workstations made it possible to store data locally. Enterprise data was distributed over networks of computer systems. To access information it was no longer necessary to have a continuous connection to central databases, and instead it was possible to copy information to a personal computer, edit and work with it, and then save it back to a file or database server later. Most microcomputers worked with data in logical chunks or files. This brought a lot of power to end users, but introduced problems in managing the large quantity of enterprise data that was no

longer stored as a unique entity in one place. For example, a file that was being edited by one user could not usually be accessed or modified by other users at the same time. It was also difficult to manage multiple copies of the same data.

Toward the end of the 1980's faster microcomputers and networks made it practical to work with enterprise data in smaller chunks than files. One example of this new technology was the development of Structured Query Language (SQL) relational databases which made it possible to divide software programs into a 'Client' tier and a 'Server' tier, that communicated with each other over a network. Client-server computing thus made it possible to store information centrally, yet manage and work with it locally. In the client-server paradigm, the client systems concentrated on offering a user-friendly interface to server data, while the server systems were able to handle many client systems at once while safely managing enterprise data.

However, the increasing client-server computing introduced its share of problems. Protocols used to communicate between client and server became increasingly complex and difficult to manage. Enterprise IT departments needed increasingly greater resources to manage the proprietary implementations of client operating systems, server database systems and middleware protocols connecting the various 'tiers' of client-server systems. Data was no longer stored in one place but was required to be managed within a distributed network of systems. Client-server systems also lacked a major advantage of mainframes: in a client-server system any changes to the data on the server weren't immediately updated on the client.

Starting in the 1990s, the Internet has allowed businesses, organizations, and other enterprises to easily make information available to users without the complex architecture that client-server systems typically require. Today, an increasing number of software applications are moving their data and logic or functional processes back to the server tier, from which they can be accessed from the Internet by a wide variety of clients, including thin and very thin-clients, which typically consist of Internet browsers or small applications (applets) whose sole responsibility is providing an interface to the user. In many ways, Internet computing (often referred to as e-commerce) has brought back the data-handling advantages of mainframes. Within the e-commerce environment data that change on the server are immediately available to clients that access the data through the Internet (world-wide) or through an intranet (enterprise-wide).

Unfortunately, the rise of Internet commerce has also given rise to some of the disadvantages associated with mainframe technology. Most Internet connections that present data to the user or client process use the Hyper Text Transfer Protocol (HTTP) which is inherently "session-less." This means that, for example, there is no totally reliable way for the server to automatically update the client display once the server data change. It also means that the server only checks the validity of the client or user input after the user sends back or submits an entire input form. This apparent disadvantage has also played an important role in the success of the Internet: because HTTP connections are session-less, they require much less processing power and much less memory on the server while the user is busy entering data. Thus, Internet applications running on web servers can be accessed by millions of people. Because HTTP and related Internet-based client-server systems do not provide continuous access to server data, systems sometimes incorporate lookup tables and pre-defined values that are cached locally. For example, a list of possible countries to be selected by a user of a web page can be sent to the user's computer when that page is first sent to the user and used thereafter for subsequent country selections. Client-server applications often pre-read the data from the server the moment an application or application window is opened, in order to present users with selection lists the moment they need them. This poses problems for data that frequently changes over time since the client system may allow users to select or enter data that is no longer valid. It also poses problems for large selection lists whose transmission to the client may take a long time.

To address this some systems incorporate a local cache of the data frequently accessed by the user. A web browser may, for example be configured to remember the last pages a user visited by storing them in a local cache file. A clear disadvantage of keeping such a local cache is that it is only useful as long as the user stays on the same client computer system. Also, the local cache may include references to web pages that no longer exist.

Some other systems with limited network bandwidth (like cell phones or personal organizers) can be deployed with built-in databases (such as dictionaries and thesauri), because it would be impractical to wait for the download of an entire database, which is needed before the data is of any use. This has the disadvantage that data stored in the device may no longer be up-to-date because it's really a static database. Also, the cost of cell phones and personal organizers is greatly increased by the need for megabytes of local storage. Another important consideration is that keeping valuable data in any local database makes it vulnerable to misuse and theft. What is needed is a mechanism that addresses these issues that allows a client-server system to retain some element of a session-based system, with its increase in performance, while at the same time offering a secure communication mechanism that requires little, if any, local storage of data.

Other attempts have been made to tackle some of the problems inherent with traditional computer system interfaces, and particularly with regard to user session administration and support. These attempts include the auto-complete function systems such as used in Microsoft Internet Explorer, the spell-as-you-go systems such as found in Microsoft Word, and the wide variety of client-server session managers such as Netopia's Timbuktu and Citrix Winframe.

Auto-Complete Functionality

Many current systems provide a mechanism to auto-complete words entered into fields and documents. This 'auto-complete' functionality is sometimes called 'type-ahead' or 'predictive text entry'. Many web browsers such as Microsoft's Internet Explorer application will automatically 'finish' the entry of a URL, based on the history of web sites visited. E-mail programs including Microsoft Outlook will automatically complete names and e-mail addresses from the address book and a history of e-mails received and sent. Auto-completion in a different form is found in most graphical user interfaces, including operating systems such as Microsoft Windows and Apple Mac OS, that present lists to the user: When the user types the first character of a list entry, the user interface list will automatically scroll down to that entry. Many software development tools will automatically complete strings entered into program source code based on a known taxonomy of programming-language dependent key words and 'function names' or 'class names' previously entered by the developer. Some cell phones and personal organizers also automatically type-ahead address book entries or words from a built-in dictionary. Auto-complete functionality facilitates easy entry of data based on prediction of what options exist for the user at a single moment in time during entry of data.

Checking as You go

More and more word processing programs (most notably Microsoft Word and certain e-mail programs) include so-called 'spell checking as you type'. These programs automatically check the spelling of words entered while the user is typing. In a way, this can be seen as 'deferred auto-complete', where the word processor highlights words after they were entered, if they don't exist in a known dictionary. These spell checking programs often allow the user to add their own words to the dictionary. This is similar to the 'history lists' that are maintained for the auto-completion of URLs in a web browser, except that in this case the words are manually added to the list of possible 'completions' by the user.

Software Component Technologies

Software component technologies have provided a measure of component generation useful in client/server systems. One of these technologies is OpenDoc, a collaboration between Apple Computer, Inc. and IBM Corporation (amongst others) to allow development of software components that would closely interact, and together form applications. One of the promises of OpenDoc was that it would allow small developers to build components that users could purchase and link together to create applications that do exactly what the users want, and would make existing 'bloatware' applications (notably Microsoft Office and Corel's WordPerfect Office/Corel Office) redundant, but the technology was dropped several years ago in favor of newer technologies such as CORBA (Common Object Request Broker Architecture), developed by the Object Management Group to allow transparent communication and interoperability between software components.

Object-oriented languages and even non-object-oriented (database) systems have used component technologies to implement technical functionality. The NeXTstep operating system from NeXT Computer, Inc. (which was later acquired by Apple Computer, Inc. and evolved into the Mac operating system Mac OS X) had an object-oriented architecture from its original beginnings, that allowed software developers to create applications based on predefined, well-tested and reliable components. Components could be 'passive' user interface elements (such as entry fields, scroll areas, tab panes etc) used in application windows. But components could also be active and show dynamic data (such as a component displaying a clock, world map with highlight of daylight and night, ticker tape showing stock symbols, graphs showing computer system activity, etc.). The NeXT operating system used object frameworks in the Objective C language to achieve its high level of abstraction which is needed for components to work well. Later, Sun Microsystems, Inc. developed the Java language specification in part to achieve the same goal of interoperability. To date, Java has probably been the most successful 'open' (operating system independent) language used to build software components. It is even used on certain web sites that allow 'Java applets' on the user's Internet browser to continuously show up-to-date information on the client system.

WebObjects, an object-oriented technology developed by Apple Computer, Inc. is an Internet application server with related development tools, which was first developed by NeXT Computer, Inc. WebObjects uses object oriented frameworks that allow distribution of application logic between server and client. Clients can be HTML-based, but can also be Java applets. WebObjects uses proprietary technology that automatically synchronizes application objects between client and server. The layer that synchronizes data objects between the client and the server is called the 'Enterprise Object Distribution' (EODistribution), part of Apple's

Enterprise Objects Framework (EOF), and is transparent to the client software components and the server software components.

Session Management

Both Netopia's Timbuktu remote access systems, and Citrix, Inc.'s Winframe terminal server product, allow some element of remote access to server applications from a client system. These products synchronize user data and server data, transparently distributing all user input to the server and return all server (display) output to the client. Timbuktu does this with very little specific knowledge about the application and operating system used. This allows it to transparently work on both Microsoft Windows and Mac OS platforms. Technologies similar to Timbuktu do exist and perform the same kind of 'screen sharing'. For example, the Virtual Network Computing (VNC) system is one example of an open source software program that achieves the same goals and also works with Linux and Unix platforms.

Citrix Winframe has taken the same idea a step further by incorporating intimate knowledge of the Microsoft Windows operating system (and its Win32 APIs) to further optimize synchronization of user input and application output on the server. It can then use this detailed knowledge of the Microsoft Windows APIs to only redraw areas of the screen that it knows will change based on a user action: for example, Winframe may redraw a menu that is pulled down by the user without needing to access the server application because it knows how a menu will work.

Software Applications

Several application providers have also built upon these technologies to provide applications and application services of use to the end-user. These applications include computer-based thesauri, on-line media systems and electronic encyclopediae.

The International Standards Organization (as detailed further in ISO 2788-1986 Documentation—Guidelines for the Establishment and Development of monolingual thesauri and ISO 5964-1985 Documentation—Guidelines for the Establishment and Development of multilingual thesauri) determines suggested specifications for electronic thesauri, and thesaurus management software is now available from numerous software vendors world-wide. However, most systems have clear limitations that compromise their user-friendliness. Most commonly this is because they use a large third-party database system, such as those from Oracle Software, Inc. or Informix, Inc. as a back-end database. This means that any thesaurus terms that are displayed to the user are fetched from the database and then presented in a user interface. If one user changes the contents of the thesaurus, other users will only notice that change after re-fetching the data. While of little concern in small or infrequently changing environments, this problem is a considerable one within larger organizations and with rapidly updated content changes, for example in media publishing applications when thesaurus terms are being linked to new newspaper or magazine articles. This type of work is usually done by multiple documentalists (media content authors) simultaneously. To avoid 'mixing up' terms linked to articles, each documentalist must be assigned a certain range of articles to 'enrich' (which in one instance may be the act of adding metadata and thesaurus terms to a document). Clearly, in these situations there is a great need for live updates of data entered by these users, but a similar need exists for all client-server database programs.

SUMMARY OF THE INVENTION

The invention provides a system that offers a highly effective solution to the aforementioned disadvantages of both

client-server and Internet systems by providing a way to synchronize the data entered or displayed on a client system with the data on a server system. Data input by the client are immediately transmitted to the server, at which time the server can immediately update the client display. To ensure scalability, systems built around the present invention can be divided into multiple tiers, each tier being capable of caching data input and output. A plurality of servers can be used as a middle-tier to serve a large number of static or dynamic data sources, herein referred to as "content engines."

The present invention may be incorporated in a variety of embodiments to suit a correspondingly wide variety of applications. It offers a standardized way to access server data that allows immediate user-friendly data feedback based on user input. Data can also be presented to a client without user input, i.e. the data are automatically pushed to the client. This enables a client component to display the data immediately, or to transmit the data to another software program to be handled as required.

The present invention can also be used to simply and quickly retrieve up-to-date information from any string-based content source. Strings can be linked to metadata allowing user interface components to display corresponding information such as, for example, the meaning of dictionary words, the description of encyclopedia entries or pictures corresponding to a list of names.

Embodiments of the present invention can be used to create a user interface component that provides a sophisticated "auto-completion" or "type-ahead" function that is extremely useful when filling out forms. This is analogous to simple, client-side auto-complete functions that have been widely used throughout the computing world for many years. As a user inputs data into a field on a form, the auto-complete function analyzes the developing character string and makes intelligent suggestions about the intended data being provided. These suggestions change dynamically as the user types additional characters in the string. At any time, the user may stop typing characters and select the appropriate suggestion to auto-complete the field.

Today's client-side auto-complete functions are useful but very limited. The invention, however, vastly expands the usefulness and capabilities of the auto-complete function by enabling the auto-complete data, logic and intelligence to reside on the server, thus taking advantage of server-side power. Unlike the client-side auto-complete functions in current use, an auto-complete function created by the present invention generates suggestions at the server as the user types in a character string. The suggestions may be buffered on a middle tier so that access to the content engine is minimized and speed is optimized.

The simple auto-complete schemes currently in popular use (such as email programs that auto-complete e-mail addresses, web browsers that auto-complete URLs, and cell phones that auto-complete names and telephone numbers) require that the data used to generate the suggestions be stored on the client. This substantially limits the flexibility, power, and speed of these schemes. The present invention, however, stores and retrieves the auto-complete suggestions from databases on the server. Using the present invention, the suggestions generated by the server may, at the option of the application developer, be cached on the middle tier or on the client itself to maximize performance.

The present invention provides better protection of valuable data than traditional methods, because the data is not present on the client until the moment it is needed, and can be further protected with the use of user authentication, if necessary.

The present invention is also useful in those situations that require immediate data access, since no history of use needs to be built on the client before data is available. Indeed, data entered into an application by a user can automatically be made available to that user for auto-completion on any other computer, anywhere in the world.

Unlike existing data-retrieval applications, server data can be accessed through a single standardized protocol that can be built into programming languages, user interface components or web components. The present invention can be integrated into and combined with existing applications that access server data. Using content access modules, the present invention can access any type of content on any server.

In the detailed description below, the present invention is described with reference to a particular embodiment named QuestObjects. QuestObjects provides a system for managing client input, server queries, server responses and client output. One specific type of data that can be made available through the system from a single source (or syndicate of sources) is a QuestObjects Service. Other terms used to describe the QuestObjects system in detail can be found in the glossary given below.

QuestObjects is useful for retrieval of almost any kind of string-based data, including the following QuestObjects Service examples:

Intranet Us

Access system for database fields (for lookup and auto-complete services)

Enterprise thesauri system.

Enterprise search and retrieval systems.

Enterprise reference works.

Enterprise address books.

Control systems for sending sensor readings to a server that responds with appropriate instructions or actions to be taken.

Internet Use

Client access to dictionary, thesaurus, encyclopedia and reference works.

Access to commercial products database.

Literary quotes library.

Real-time stock quote provision.

Access to real-time news service.

Access to Internet advertisements.

Access to complex functions (bank check, credit card validation, etc).

Access to language translation engines.

Access to classification schemes (eg, Library of Congress Subject Headings).

Access to lookup lists such as cities or countries in an order form.

Personal address books.

Personal auto-complete histories.

BRIEF DESCRIPTION OF THE FIGURES

FIG. **1** shows a general outline of a system incorporating the present invention.

FIG. **2** shows a schematic of a system in accordance with an embodiment of the invention.

FIG. **3**A shows a variety of stages in the usage of a sample Questlet implementation in accordance with an embodiment of the invention.

FIG. **3**B shows an expanded view of a sample Questlet implementation in accordance with an embodiment of the invention.

FIG. **3**C shows an expanded view of a sample Questlet implementation in accordance with an embodiment of the invention.

FIG. **4** shows a sequence diagram illustrating the use of a system in accordance with an embodiment of the invention.

FIG. **5**A shows a first thread flow chart illustrating the interface between an active component and an embodiment of the invention.

FIG. **5**B shows a second thread flow chart illustrating the interface between an active component and an embodiment of the invention.

FIG. **6**A shows a first thread flow chart illustrating the client side of an embodiment of the invention.

FIG. **6**B shows a second thread flow chart illustrating the client side of an embodiment of the invention.

FIG. **7**A shows a first thread flow chart illustrating the server side of an embodiment of the invention.

FIG. **7**B shows a second thread flow chart illustrating the server side of an embodiment of the invention.

FIG. **8**A shows an object model of an embodiment of the present invention, displaying the base part.

FIG. **8**B shows an object model of an embodiment of the present invention, displaying the client part.

FIG. **8**C shows an object model of an embodiment of the present invention, displaying the server part.

FIG. **8**D shows an object model of an embodiment of the present invention, displaying the service part.

FIG. **9** shows a schematic of an application proxy system that enables the use of the invention in various client environments.

DETAILED DESCRIPTION

Roughly described, the invention provides a session-based bi-directional multi-tier client-server asynchronous information database search and retrieval system for sending a character-by-character string of data to an intelligent server that can be configured to immediately analyze the lengthening string character-by-character and return to the client increasingly appropriate database information as the client sends the string.

The present invention includes a system that offers a highly effective solution to an important disadvantage of both client-server and Internet systems: The present invention provides a standardized way to immediately synchronize the data entered or displayed on a client system with the data on a server system. Data input by the client is immediately transmitted to the server at which time the server can immediately update the client display. To ensure scalability, systems built around the present invention can be divided into multiple 'tiers' each capable of caching data input and output. Any number of servers can be used as a middle-tier to serve any number of static or dynamic data sources (often referred to as "Content Engines").

The present invention is useful for an extremely wide variety of applications. It offers a standardized way to access server data that allows immediate user-friendly data feedback based on user input. Data can also be presented to a client without user input, i.e. the data is automatically 'pushed' to the client. This enables a client component to display the data immediately or to transmit it to another software program to be handled as required.

The present invention is also particularly useful for assistance in data entry applications, but can also be used to simply and quickly retrieve up-to-date information from essentially any string-based content source. Strings can be linked to metadata allowing user interface components to display corresponding information such as the meaning of dictionary words, the description of encyclopedia entries or pictures corresponding to a list of names.

In some embodiments, the present invention can be used to create a user interface component that provides a sophisticated "auto-completion" or "type-ahead" function that is extremely useful when filling out forms. Simple, client-side auto-complete functions have been widely used throughout the computing world for many years. As a user inputs data into a field on a form, the auto-complete function analyzes the developing character string and makes "intelligent" suggestions about the intended data being provided. These suggestions change dynamically as the user types additional characters in the string. At any time, the user may stop typing characters and select the appropriate suggestion to auto-complete the field.

Today's client-side auto-complete functions are very limited. The present invention vastly expands the usefulness and capabilities of the auto-complete function by enabling the auto-complete data, logic and intelligence to reside on the server thus taking advantage of server-side power. Unlike the client-side auto-complete functions in current use, an auto-complete function created by the present invention pushes suggestions from the server as the user types in a character string. Using the present invention, the suggestions may be buffered on a middle tier so that access to the content engine is minimized and speed is optimized.

The simple auto-complete schemes currently in popular use (such as email programs that auto-complete e-mail addresses, web browsers that auto-complete URLs, and cell phones that auto-complete names and telephone numbers) require that the data used to generate the suggestions be stored on the client. This substantially limits the flexibility, power, and speed of these schemes. The present invention, however, stores and retrieves the auto-complete suggestions from databases on the server. Using the present invention, the suggestions generated by the server may, at the option of the application developer, be cached on the middle tier or one the client itself to maximize performance.

The present invention provides better protection of valuable data because the data is not present on the client until the moment it is needed and can be further protected with a user authentication mechanism, if necessary.

The present invention is useful for immediate data use, since no use history must be built on the client before data is available. Indeed, data entered into an application by a user can automatically be made available to that user for auto-completion on any other computer anywhere in the world.

Unlike existing data-retrieval applications, server data can be accessed through a single standardized protocol that can be built into programming languages, user interface components or web components. The present invention can be integrated into, and combined with, existing applications that access server data. Using Content Access Modules, the present invention can access any type of content on any server.

In the detailed description below, an embodiment of the present invention is referred to as QuestObjects, and provides a system of managing client input, server queries, server responses and client output. One specific type of data made available through the system from a single source (or syndicate of sources) is referred to as a QuestObjects Service. Other terms used to describe the QuestObjects system in detail can be found in the glossary below:

GLOSSARY

Active Component—Part of a software program that accesses the QuestObjects system through one or more Questers. Active Components may provide a user interface, in which case they're referred to as Questlets.

AppHost Synchronizer—Part of the QuestObjects Server that allows the Application Proxy access to data in Server Questers.

Application Proxy—An optional method implemented by the QuestObjects Server allowing the use of the QuestObjects system in client systems that do not allow the

QuestObjects—Client components to communicate to the application server or web server directly. Uses the AppHost Synchronizer on the QuestObjects Server to send selected strings and metadata to the application server or web server using a QuestObjects Adaptor.

Client Controller—A QuestObjects Controller on a QuestObjects Client.

Client Quester—A Quester on a QuestObjects Client that has a Server Quester as its peer.

Client Session—A temporary container of information needed to manage the lifespan of Server Questers in a QuestObjects Server.

Content Access Module—A part of a Content Channel that provides a standardized API to access specific types of Content Engines.

Content-based Cache—A persistent store of Queries and corresponding Result Sets executed by a Content Engine for a specific Content Channel.

Content Channel—A part of the QuestObjects system that provides one type of information from one Content Engine. Consists of a Query Manager and a Content Access Module, linking a Content Engine to the QuestObjects system.

Content Engine—A dynamic data source that provides data to a Content Channel by accessing its own database or by querying other information systems.

Query Filter—A filter specified by a Query Manager in a specific Service used to tell the Server Quester to interpret incoming strings before they are sent to the Service as a QuestObjects Query.

Query Manager—An intelligent part of a Content Channel that interprets QuestObjects Queries and sends them to a Content Engine (through a Content Access Module) or retrieves results from the Content-based Cache in a standardized way. The Query Manager can also send a list of Query Patterns and Query Filters to the Server Quester, allowing the Server Quester to match and filter new Queries before they are sent to the Content Channel.

Query Pattern—A string-matching pattern (such as a unix-style grep pattern) specified by a Query Manager in a specific Service used to tell the Server Quester to interpret incoming strings before they are sent to the Service as a QuestObjects Query.

Persistent Quester Store—A dynamic database of Questers that is maintained on the QuestObjects Server, allowing Questers to be stored across Client sessions whereby the state and contents of the Client are automatically restored when a new Client Session is started.

Quester—An intelligent non-visual object contained by an Active Component that links a QuestObjects StringList to an input buffer. Questers exist on both the QuestObjects Client and the QuestObjects Server and can be specifically referred to as Client Quester and Server Quester. Questers communicate with each other through a QuestObjects Controller.

Questlet—A User Interface Element that accesses the QuestObjects system through one or more Questers. A visual Active Component.

QuestObjects Adaptor—An optional software component for existing application servers and web servers that allows these servers to use data entered into the QuestObjects system by users of client systems and web browsers that require an Application Proxy.

QuestObjects Client—Part of the QuestObjects system that functions as the client tier consisting of one or more Client Questers and a Client Controller that communicates to a QuestObjects Server.

QuestObjects Controller—An intelligent non-visual component that provides the interface between Questers on QuestObjects Clients and QuestObjects Servers. QuestObjects Controllers implement the protocol of the present invention.

QuestObjects Query—A string created by the Server Quester with optional qualifier and the requested row numbers forming a query to be executed by a specified QuestObjects Service.

QuestObjects Result Set—A set of StringLists with corresponding Query returned from the QuestObjects Service, returned in batches to the Client Quester by the Server Quester.

QuestObjects Server—Central part of the QuestObjects system that provides the link between any number of QuestObjects Clients, any number of QuestObjects Services, and any number of other QuestObjects Servers. Maintains Client Sessions that QuestObjects Clients communicate with through the Server Controller. Provides services such as caching, replication and distribution.

QuestObjects Service—One of the Content Channels provided by a specific Syndicator. A logical name for a Syndicator, a Content Channel and its corresponding Content Engine.

QuestObjects String—Sequence of Unicode characters with standardized attributes used by the QuestObjects system.

QuestObjects StringList—Container for a set of QuestObjects Strings retrieved from a QuestObjects Service with standardized attributes needed by the QuestObjects System.

QuestObjects User—Person or process accessing the QuestObjects system from the QuestObjects Client, optionally authorized by the Syndicator.

Server Controller—A QuestObjects Controller on a QuestObjects Server.

Server Quester—A Quester on a QuestObjects Server that has a Client Quester as its peer.

Syndicator—A part of the QuestObjects system that offers one or more Content Channels to be used by QuestObjects Servers, performing user-based accounting services based on actual data use such as billing, collection of statistics and management of preferences.

User Interface Element—A visual and optionally interactive component in a software program that provides an interface to the user.

The present invention provides a system that allows clients or client applications to asynchronously retrieve database information from a remote server of server application. The terms "client" and "server" are used herein to reflect a specific embodiment of the invention although it will be evident to one skilled in the art that the invention may be equally used with any implementation that requires communication between a first process or application and a second process or application, regardless of whether these processes comprise a typical client-server setup or not. The invention includes a Server, that handles requests for information from clients, and a communication protocol that is optimized for sending single characters from a Client to the Server, and lists of strings from the Server to the Client. In one embodiment, as the Server receives a single character from the Client, it immediately analyzes the lengthening string of characters and, based on that analysis, returns database information to the Client in the form of a list of strings. Clients are not restricted to programs with a user interface. Generally, any process or mechanism that can send characters and receive string lists can be considered a client of the system. For example, in an industrial or power supply setting, the control system of a power plant could send sensor readings to the system, and in return receive lists of actions to be taken, based on those sensor readings.

The system's protocol is not restricted to sending single characters. In fact, Clients can also use the protocol to send a string of characters. For example, when a user replaces the contents of an entry field with a new string, the Client may then send the entire string all at once to the Server, instead of character by character.

In accordance with one embodiment of the invention the system is session-based, in that the server knows or recognizes when subsequent requests originate at the same Client. Thus, in responding to a character the Server receives from a Client it can use the history of data that has been sent to and from the current user. In one embodiment, the system stores user preferences with each Service, so that they are always available to the Client, (i.e., they are independent of the physical location of the client). Furthermore, client authentication and a billing system based on actual data and content use by Clients are supported. For faster response, the Server may predict input from the Client based on statistics and/or algorithms.

The system is bi-directional and asynchronous, in that both the Client and the Server can initiate communications at any moment in time. The functionality of the system is such that it can run in parallel with the normal operation of clients. Tasks that clients execute on the system are non-blocking, and clients may resume normal operation while the system is performing those tasks. For example, a communication initiated by the Client may be a single character that is sent to the Server, that responds by returning appropriate data. An example of a communication initiated by the Server is updating the information provided to the client. Because the system is session-based it can keep track of database information that has been sent to the Client. As information changes in the database, the Server sends an updated version of that information to the Client.

Embodiments of the system may be implemented as a multi-tier environment This makes it scalable because the individual tiers can be replicated as many times as necessary, while load balancing algorithms (including but not limited to random and round robin load-balancing) can be used to distribute the load over the copies of the tiers. One skilled in the art would appreciate that it is not necessary to replicate the tiers. Indeed, there may be only a single copy of each tier, and that all tiers (Client, Server, and Service) may be running on a single computer system.

FIG. **1** illustrates the general outline of a system that embodies the present invention. As shown in FIG. **1** there may be various Clients **101** using the system. These Clients use a communication protocol **102** to send information, including but not limited to single characters, and to receive information, including but not limited to lists of strings and corresponding metadata. At least one Server **103** receives information from the Client, and sends information to the Client. In a typical embodiment if there is a plurality of Servers, then the system can be designed so that each Client connects to only one of them, which then relays connections to other Servers, possibly using load-balancing algorithms. Servers have a communication link **104** to a Service **105**, which they use to obtain the information that they send to the Client.

FIG. **2** is a schematic illustrating an embodiment of the present invention, and displays a five-tier system that has a user interface in which user interface elements use the present invention to assist the user in performing its tasks. For purposes of illustration, FIG. **2** displays just one session and one content Service. In an actual implementation there may be multiple concurrently active sessions, and there may be more than one content Service that Clients can use. As shown herein, the first of the five tiers is a Client tier **201**. The Client tier contains the user interface and the Client components that are needed to use the system. The second tier is a Server or server process **206**, which handles the queries that Clients execute, and in return displays results to the Client. Service **213**, which corresponds to **105** of FIG. **1**, is a logical entity consisting of three more tiers: a Syndicator **214**, a Content Channel **219** and a Content Engine **224**. The Syndicator provides access to a number of Content Channels and performs accounting services based on actual data use. The Content Channel provides a specific type of information from a specific source (i.e. the Content Engine). The Content Engine is the actual source of any content that is made available through the QuestObjects system. The Client tier **201** corresponds to the client **101** in FIG. **1**. In this example, the Client may be an application (and in some embodiments a web application) with a user interface that accesses the system of the present invention. As used in the context of this disclosure a user interface element that uses the present invention is referred to as a "Questlet." A Client can contain one or more Questlets **202** (e.g. an input field or a drop down list. FIG. **3** described later contains three examples of such Questlets. A Questlet is always associated with at least one Client Quester **203**. Questers are objects that tie a QuestObjects input buffer (containing input from the Client) to a QuestObjects Result Set returned from a QuestObjects Server. Questers exist on both the Client and Server, in which case they are referred to as a Client Quester and a Server Quester, respectively. Every Client Quester has one corresponding Server Quester. In accordance with the invention, any event or change that happens in either one of them is automatically duplicated to the other so that their states are always equal. This synchronization mechanism is fault-tolerant so that a failure in the communication link does not prevent the Questers from performing tasks for which they do not need to communicate. For example, a Client Quester can retrieve results from the cache, even if there is no communication link to the Server. Each single Quester accesses exactly one QuestObjects Service, i.e. one specific Content Channel offered by one specific Syndicator. At initialization of the Client, the Questlet tells its Quester which Service to access. In one embodiment a Service is stored or made available on only one Server within a network of Servers. However, this is transparent to the Client because each Server will forward requests to the right computer if necessary. The Client does not need to know the exact location of the Service.

To communicate with its Server Quester **208**, each Quester in a session uses a controller **204**. The system contains at least one Client Controller **204** and a Server Controller **209**, which together implement the network communication protocol **205** of the present invention. Client Controllers may cache results received from a Server, thus eliminating the need for network traffic when results are reused.

Client Questers are managed by a Questlet, which create and destroy Questers they need. In a similar fashion, Server Questers are managed by a Session **207**. When a Client Quester is created, it registers itself with the Client Controller. The Client controller forwards this registration information as a message to the Session using the Server Controller. The Session then checks if the Persistent Quester Store **210** contains a stored Quester belonging to the current user matching the requested Service and Query Qualifier. If such a Quester exists, it is restored from the Persistent Quester Store and used as the peer of the Client Quester. Otherwise, the Session creates a new Server Quester to be used as the Client Quester's peer.

A Time Server **211** provides a single source of timing information within the system. This is necessary, because the system itself may comprise multiple independent computer systems that may be set to a different time. Using a single-time source allows, for example, the expiration time of a Result Set to be calibrated to the Time Server so that all parts of the system determine validity of its data using the same time.

Server communication link **212** is used by the Server to send requests for information to a Service, and by a Service to return requested information. Requests for information are Query objects that are sent to and interpreted by a specific Service. Query objects contain at least a string used by the Service as a criterion for information to be retrieved, in addition to a specification of row numbers to be returned to the Client. For example, two subsequent queries may request row numbers 1 through 5, and 6 through 10, respectively. A query object may also contain a Qualifier that is passed to the appropriate Service. This optional Qualifier contains attributes that are needed by the Service to execute the Query. Qualifier attributes may indicate a desired sort order or in the example of a thesaurus Service may contain a parameter indicating that the result list must contain broader terms of the Query string. Services use the communication link to send lists of strings (with their attributes and metadata) to Servers. Server communication link **212** is also used by Server Questers to store and retrieve user preferences from a Syndicator's Preference Manager.

Questers use Services to obtain content. A Service is one of the Content Channels managed by a Syndicator. When a Quester is initialized, it is notified by its Active Component of the Service it must use. The Service may require authentication, which is why the Syndicator provides a User Manager **215**. If a Client allows the user to set preferences for the Service (or preferences needed by the Active Component), it may store those preferences using the Syndicator's Preference Manager **216**. The Server (i.e. Server Quester) only uses the Syndicator for authentication and preferences. To obtain content, it accesses the appropriate Content Channel directly. The Content Channel uses its Syndicator to store usage data that can be later used for accounting and billing purposes. Usage data is stored in a Usage Statistics Store **217**.

Content communication link **218** is used by Content Channels to send usage data to their Syndicator, and to retrieve user information from the Syndicator. The Content Channel is a layer between the QuestObjects System, and the actual content made available to the system by a Content Engine **224**. Each Content Channel has a corresponding Query Manager **220** that specifies the type of query that can be sent to the corresponding Content Engine, and defines the types of data that can be returned by the Content Channel.

Specification of query type comprises a set of Query Patterns and Query Filters that are used by the Server Quester to validate a string before the string is sent to the Content Channel as a QuestObjects Query. For example, a query type "URL" may allow the Server Quester to check for the presence of a complete URL in the input string before the input string is sent to the Content Channel as a query. A query type "date" might check for the entry of a valid date before the query is forwarded to the Content Channel.

The Query Manager optionally defines the types of string data that can be returned to the Client by the Content Channel. Specific Active Components at the Client can use this information to connect to Services that support specific types of data. Examples of string types include: simple terms, definitional terms, relational terms, quotes, simple numbers, compound numbers, dates, URLs, e-mail addresses, preformatted phone numbers, and specified XML formatted data etc.

The Query Manager **220** retrieves database information through a Content Access Module **221**. The Content Access Module is an abstraction layer between the Query Manager and a Content Engine. It is the only part of the system that knows how to access the Content Engine that is linked to the Content Channel. In this way, Query Managers can use a standardized API to access any Content Engine. To reduce information traffic between Content Channels and Content Engines, Content Channels may access a content-based cache **222** in which information that was previously retrieved from Content Engines is cached. Engine communication link **223** is used by Content Access Modules to communicate with Content Engines. The protocol used is the native protocol of the Content Engine. For example, if the Content Engine is an SQL based database system then the protocol used may be a series of SQL commands. The Content Access Module is responsible for connecting the Content Engine to the QuestObjects System.

Content Engines **224** are the primary source of information in the system. Content Engines can be located on any physical computer system, may be replicated to allow load balancing, and may be, for example, a database, algorithm or search engine from a third-party vendor. An example of such an algorithm is Soundex developed by Knuth. Content Engines may require user authentication, which, if required, is handled by the Syndicator (through the Content Access Module).

The invention uses Content Engines as a source of strings. One skilled in the art would understand that a string may, for example, contain a URL of, or a reference to any resource, including images and movies stored on a network or local drive. Furthermore, strings may have metadata associated with them. In one embodiment, strings might have a language code, creation date, modification date, etc. An entry in a dictionary may have metadata that relates to its pronunciation, a list of meanings and possible uses, synonyms, references, etc. A thesaurus term may have a scope note, its notation, its source and its UDC coding as metadata, for example. Metadata of an encyclopedia entry may include its description, references, and links to multi-media objects such as images and movies. A product database may have a product code, category, description, price, and currency as metadata. A stock quote may have metadata such as a symbol, a company name, the time of the quote, etc. Instructions to a control system may contain parameters of those instructions as metadata. For example, the instruction to open a valve can have as metadata how far it is to be opened.

FIGS. **3A-3C** contain three examples of the Questlets that can be used with the system, i.e., the User Interface Elements that access the QuestObjects system. In FIG. **3A**, a series of representations of an auto-completing entry field are shown, such as might be used in an application window or on a web form, that accesses a single QuestObjects Service, and allows for auto-completion of, in this example, a U.S. state name. FIGS. **3B** and **3C** depict two different presentation forms of the same complex Questlet that access a number of QuestObjects Services simultaneously.

Users should be able to clearly recognize the availability of QuestObjects Services in an application. As shown in FIG. **3A**, and particularly in the auto-complete entry field example screen element **302**, clear symbols are displayed at the right end of the field. A small disclosure triangle **308** is displayed in the lower right-hand corner, and serves as an indicator to the user that a QuestObject is being used. A reserved space herein referred to as the "status area", and located above the disclosure triangle **301** is used to display information about the state of the QuestObjects system. The successive shots of this screen element **302** through **307** show some of the different kinds of states in this status area. Screen element **302** depicts an empty data field with an empty status area. The screen element **303** shows the same field immediately after the user enters a character "N". On receiving the "N"input, the Questlet immediately checks its internal entry cache for available auto-complete responses. If the cache does not contain a valid string (either because the cache is empty, because the cache is incomplete for the entry character, or because one or more cached strings have expired) the QuestObjects system sends a query to the QuestObjects Service. This sending process is indicated by a network access symbol in the status area **304** which is in this embodiment takes the form of a left and right facing arrows.

Screen element **305** shows the entry field after the Server has sent one or more auto-complete strings back to the Questlet. This example situation is typical of these instances in which the user did not enter a second character after the original "N" before the QuestObjects system responded. The QuestObjects system is inherently multi-threaded and allows the user to continue typing during access of the QuestObjects Service. The screen element status area of **305** now displays a small downward facing arrow indicating that there are more available auto-complete answers. In this case, the entry field has displayed the first one in alphabetic order.

Screen element **306** shows the same entry field after the user has hit the down arrow key or clicked on the arrow symbol in the status area. The next available auto-complete response in alphabetical order is displayed. The double up and down pointing arrows in the status area now indicate that both a previous response (in this example, "Nebraska") and a next response are available.

Screen element **307** shows the same entry field after the user has typed two additional characters, "e" and "v". As shown in this example, the status area changes to a checkmark indicating that there is now only one available auto-complete match for the characters entered. The user can at any point use the backspace key on their keyboard (or perform other actions defined in the Questlet) to select different states, or can leave the entry field to confirm his selection. At this time, the system may do several things. It can automatically accept the string "Nevada" and allow the user to move on with the rest of the entry form; or if it has been configured such it may decide to replace the string "Nevada" by the two-character state code. The QuestObjects Service not only returns strings, but also any corresponding metadata. This example of an auto-complete entry field Questlet is based on showing the response string, but other Questlets (and even invisible Active Components) may perform an action invisible to the user. In addition, a response sent to one Questlet can trigger a response in other Questlets that have a pre-defined dependency to that Questlet. For example, entering a city into one Questlet can trigger another Questlet to display the corresponding state. It will be evident to one skilled in the art, that although left, right, up and down arrows are used to indicate usually the status of the QuestObject field, other mechanisms of showing the status within the scope and spirit of the invention.

Interdependent data (which in the context of this disclosure is that data originating from a multitude of QuestObjects Services) can be combined into a complex Questlet. Examples **309** shown in FIG. **3B** and example **313** shown in FIG. **3C** show a complex user interface element (Questlet) that makes multiple QuestObjects Services available to the user. In both examples the upper part of the Questlet is an entry field that may offer the auto-complete functionality described in FIG. 3A. By clicking on the disclosure triangle **308** shown in the earlier FIG. 3A (or by another action), the user can disclose the rest of the Questlet, which in this example comprises two functional areas **311** and **312**. In this example, the user interface allows the user to choose a vertical presentation mode **309**, shown in FIG. 3B or a horizontal presentation mode **313**, shown in FIG. 3C for the Questlet. A close box **310** replaces the disclosure triangle in the entry field, allowing the user to close areas **311** and **312**. In FIG. 3C Area **314** shows a certain QuestObjects Service, in this case a list of "Recent Terms" accessed by the user. This Questlet allows the user to select a different QuestObjects Service for area **314** by selecting it from a popup list **319**. In this example, an appropriate second Service might be "Alphabetic Listing".

In both examples of FIGS. 3B and 3C, area **312** displays a QuestObjects "Thesaurus Service" (Thesa) that has been selected. Additionally, in FIG. 3C areas **315** through **318** display four different Questers that take their data from a QuestObjects Thesaurus Service. These Questers all access the same Thesaurus and all have a dependency on the selected string in the main list of area **314**. Once the user clicks on a string in area **314** the thesaurus lists **315** through **318** are automatically updated to show the corresponding "Used For terms" UF, "Broader Terms" BT, "Narrower Terms" NT, and "Related Terms" RT from the Thesaurus Service. Questers **315** through **318** thus have a different Qualifier that is used to access the same QuestObjects Service. It will be evident to those skilled in the art that this example is not intended to be a complete description of features that a thesaurus browser (or any other Service) provides. Most thesauri offer a multitude of term relationships and qualifiers. A Questlet or part of a Questlet may provide access to a multitude of QuestObjects Services. A possible way to do this is to show multiple tabbed panes accessible through tab buttons named after the Services they represent **320**.

Data from the QuestObjects Services can be displayed by a Questlet in many forms. Thesaurus browser Questlets generally display interactive lists of related terms. Questlets can also allow users to lookup data in a reference database (dictionary, encyclopedia, product catalog, Yellow Pages, etc.) made available as a QuestObjects Service. Furthermore, Questlets can access QuestObjects Services that provide a standardized interface to search engines. These search engines may be Internet-based or can be built into existing database servers. Questlets can also access pre-defined functions made available as QuestObjects Services (such as a bank number check, credit card validation Service or encryption/decryption Service). Questlets can even access translation Services allowing on-the-fly translation of entry data. In some embodiments Questlets can retrieve multi-media data formats by receiving a URL or pointer to multi-media files or streaming media from a QuestObjects Service. In other embodiments Questlets can be used to display current stock quotes, news flashes, advertisements, Internet banners, or data from any other real-time data push Service. Questlets can provide an auto-complete or validity checking mechanism on the data present in specific fields or combinations of fields in relational database tables.

As described above, Questlets are well suited to represent QuestObjects data visually. However, a QuestObjects Client system can also contain non-visual Active Components, such as function calls from within a procedure in a program to access a QuestObjects Service. A program that needs to display a static or unchanging list of strings can use a Quester in its initialization procedure to retrieve that list from a QuestObjects Server. By calling a Quester, a stored procedure in a database can make a QuestObjects Service available to any database application. By encapsulating a Quester into an object supplied with a programming language, a QuestObjects Service can be made available to its developers. Another example of how QuestObjects Services may be accessed is through a popup menu that a user can access by clicking on a word, phrase or sentence in a document. The popup menu can include one or more QuestObjects Services by calling one or more Questers. In an application that is controlled by speech, a sound conversion engine that translates speech input into phonemes can be used to send these phonemes to a QuestObjects speech recognition Service through a Quester. As yet another example, a control system can use a Quester to send sensor readings to a Server, which then queries a special purpose content engine to return actions that the control system must perform given the sensor readings.

FIG. **4** shows a simplified event life cycle illustrating what happens in a QuestObjects system using an auto-complete Service. The protocol of the present invention is implemented in the Client Controller and the Server Controller **400**. In an initial phase an Active Component on the Client tells its Quester to start or initialize **401** a corresponding Client Session on the current QuestObjects Server by sending a Register message to its Client Controller. The Server Controller starts a Client Session if it has not been started already. For simplicity the event trace of FIG. **4** does not show typical error handling that normally occurs, for instance when a Session cannot be started. If the Quester was used before in the same Active Component and application, the Session may restore the Quester from a Persistent Quester Store, which may even cause a Query to be triggered immediately if the Result Set in the Quester is out of date.

The Server Quester looks up the Service in the Server's list of known QuestObjects Services, which may or may not be located on the same computer. Once the Service is found, the Client is registered and optionally authenticated by the Service. At this time, the Service **402** returns information to the Server Controller at which time the Client receives a confirmation that it was registered successfully. The Active Component can now start using the Quester it has just initialized. If the Active Component has a user interface (i.e. it is a Questlet) then it will now allow the user to start entering characters or cause other user events.

The next step in the process is to capture user input. As shown in FIG. **4**, at point **403** a character event is generated to indicate the user has typed a character 'a' into the Questlet. The Quester sends a message to its Client Controller telling it that character 'a' must be appended to the input buffer (it will be evident to one skilled in the art that if the cursor is not at the end of the input string, typing 'a' would, for example, generate a different event to insert the character instead of append it). The Client Controller uses the protocol to synchronize the input buffer in the Server Quester by communicating to the Server Controller. The Server Controller may look up query 'a' in its Result Set cache, in which case it can return a previous Result Set to the Client without accessing the Service. Also, depending on any rules specified by the Service (as specified by a list of Query Patterns and Query Filters defined in the Query Manager of the Content Channel) and depending on the time interval between input buffer changes, the Server Quester may decide not to immediately send the (perhaps incomplete) string to the Service, as shown here.

An additional character event **404** is generated when the user has typed a second character 'b' into the Questlet. As before, a corresponding event arrives at the Server Quester. In this case, the Server Quester may deduct that the input string represents a valid query and send the appropriate query message 'ab' to the Service. After receiving a query, the Service executes it by accessing its Content Engine through the Content Access Module unless the Query Manager was able to lookup the same Query with a Result Set in the Content-based Cache. After an appropriate Result Set **405** is retrieved, the Service will return it to the Client. In some embodiments, a large Result Set may be returned to the Client in small batches. In other embodiments an incomplete Result Set may also be returned if the Content Engine takes a long time to come up with a batch of results. A QuestObjects Service may automatically 'push' updated information matching the previous query to the Client as it becomes available. A Query can also be set to auto-repeat itself **406** if necessary or desired.

At step **407** the user types a third character 'c' into the Questlet. While this character is being sent to the Server, a second and possibly third result set from the previous query is on its way to the Client. When the Client Controller decides **408** that the received Result Set 'ab' no longer matches the current input string 'abc', the second update of 'ab' is not transmitted to the Active Component. Depending on the sort order and sort attributes of the Result Set, the Client Controller may still send the second and third Result Sets to the Active Component if the second query 'abc' matches the first string of the Result Set for the first query 'ab' **409**. In that case, the user typed a character that matched the third character in the second or third Result Set, thus validating the Result Sets for the second query. Eventually the Server Quester receives notice of the third character appended to the input buffer, and sends a new query 'abc' to the Service. The Server Quester will stop the 'repeating' of query 'ab' and the Service will now execute **410** the new query 'abc' at the Content Engine, or retrieve it from the Content-based Cache.

FIG. **5** depicts a flow chart illustrating the interface between an Active Component and the present invention. As shown therein a Client Quester is initialized (step **501**) in which each active component is associated with one or more Client Questers. A loop is then entered that exits when the Active Component is destroyed (step **502**). In the loop, events are sent to the Client Quester (step **503**), such as keyboard events, click events and focus events (i.e. events that tell the system which user interface element currently has input focus). When events are sent to the Client Quester, they may result in return events from the Client Quester, such as events informing that the Result Set of the Client Quester has changed. Those events are received by the event receiver (step **504**). The event receiver waits for events from the Client Quester (step **506**) and—if events have been received (**507**)—processes them (step **508**). It will be evident to one skilled in the art that the Active Component can be multi-threaded, in that the event receiver can work concurrently with the rest of the Active Component. The Active Component may also use a cooperative multi-threading scheme where it actively handles client events and server responses in a continuous loop.

FIG. **6** shows a flow chart illustrating the Client side of the present invention. First, the Client Quester registers itself with the Client Controller (step **601**). It then enters a loop that exits when the Client Quester is destroyed (step **602**). When that happens, the Client Quester deregisters itself from the Client Controller (step **603**). During the loop the Client Quester handles events from the Active Component it belongs to. First, it waits for an event and receives it (step **604**). Then the type of the event is checked (step **605**). If it is not a character event, it is handled depending on the type and content of the event (step **606**). An example of a non-character event is a double-click on the input string, the click of a button

that clears the input buffer, the addition of characters to the input buffer by a paste-action etc. If the event is a character event, the input buffer is updated accordingly and Client Questers that have dependencies with the input buffer or the Result Set also are notified (step **607**).

The next step is to get results based on the new input buffer. First, the Client Quester checks if the results are present in the client-side cache, which usually is a fast short-term in-memory buffer (step **608**); if so, they are retrieved from the cache (step **609**) and the Active Component is notified of the results (step **610**). If the results are not found in the cache, the Client Quester uses the Client Controller to send the new input buffer to the Server Quester, so that a new query can be executed (step **611**). To support this, the protocol of the present invention provides a number of messages that allow the Client Quester to send just the changes to the input buffer, instead of sending the entire input buffer. These messages include but are not limited to: inputBufferAppend, inputBufferDeleteCharAt, inputBufferInsertCharAt, inputBufferSetCharAt, inputBufferSetLength, and inputBufferDelete. After thus updating the Server Quester's input buffer, the Client Quester activates the result retriever to wait for new results and process them (step **612**).

The Client Quester is intended to be multi-threaded, so that it can continue providing its services to its Active Component while it waits for results from the QuestObjects Server. Therefore, the Result Retriever can be implemented to run in a separate thread of execution. In this embodiment the Result Retriever waits for results from the Server Quester (step **613**). If results have been received (step **614**), it checks whether they are usable (step **615**). Results are usable if they correspond to the latest query. If results are from a previous query (which can occur because the system is multi-threaded and multi-tier), they may also still be usable if the Client Quester can filter them to match the new input buffer (this depends on the sort flags in the Result Set). If results are usable, the Active Component is notified of the new results. This notification is also sent to other Client Questers that have dependencies on the originating Client Quester (step **616**). Received results are stored in the client-side cache, regardless of whether they were found to be usable (step **617**).

FIG. **7** is a flow chart illustrating the Server side of the present invention. The first thing a Server Quester does when it is created, is to check whether its attributes can be restored from the Persistent Quester Store (step **701**), based on the parameters with which it is created. If the attributes can be restored, they are restored and registered with its corresponding Service (step **702**). In accordance with one embodiment, one of the restored attributes is a Result Set attribute; the Server Quester checks whether it is still up to date (step **703**). If not, a query is sent to the corresponding Service if it is a pushing service or if the Query was originally set to be auto-repeating (step **704**) and (in a separate thread of execution) the Server Quester waits for the results of that query and processes them (step **705**).

If the Server Quester's attributes could not be restored, it initializes itself and registers itself with the correct service which is one of the initialization parameters (step **706**). If the Client Quester was created with a default input buffer, the Server Quester may automatically send the corresponding Query to the Service. At this point, the initialization process is complete and the Server Quester enters a loop that exits when the Quester is destroyed (step **707**). During the loop, the Server Quester checks whether the Query String is valid, using the validation attributes of the Service (Query Pattern and Query Filter) (step **708**). If the query is valid, the Server Quester checks if the server-side cache has the results for the

Query String (step **709**). If not, a new Query is sent to the Service (step **710**). After that, the results are retrieved (either from cache or from the Service) and processed (step **711**).

After validating (and possibly processing) the Query String, the Server Quester waits for messages from the Client Quester notifying of changes to the input buffer (step **712**). If such a message is received, the input buffer is updated accordingly (step **713**), and the loop is re-entered (step **708**).

The processing of query results is performed in a separate thread of execution. The process performed in this thread starts by obtaining the Result Set (step **714**), either from the server-side cache or from the Service depending on the result of the decision in step **709**. When these results are obtained (step **715**), they are sent to the Client Quester (step **716**) either as part of the Result Set or as the entire Result Set, depending on parameters set by the Client Quester and are stored in the server-side cache (step **717**). In addition, the Service is notified of actual results that have been sent to the client (step **718**). If the results were pushed by the Service (step **719**), this thread starts waiting for new results to be processed; otherwise, the thread stops.

FIGS. **8A**-**8D** illustrate and object model of an embodiment of the present invention. FIG. **8A** illustrates the base portion of the model containing the entities that are not specific to either QuestObjects Clients, QuestObjects Servers, or QuestObjects Services. FIG. **8B** displays the entities that are specific to the QuestObjects client. FIG. **8C** contains the entities specific to the QuestObjects Server. FIG. **8D** shows the entities specific to the QuestObjects Service.

Each of FIGS. **8A** through **8D** show object models of one particular embodiment of the present invention, using UML (Unified Modelling Language) notation. Note that in the figures some of the entities have a name that starts with one of the words 'base', 'client', 'server', and 'service', followed by two colons. Those entities are merely references to entities in the subfigure indicated by the word before the two colons. For example, the entity named 'service::QoService' in FIG. **8A** is a reference to the 'QoService' entity in the figure of the service part, namely FIG. **8D**. It will be evident to one skilled in the art that the model shown is purely an illustrative example of one embodiment of the invention and that other models and implementations may be developed to practice the invention while remaining within the spirit and scope of the this disclosure.

The base part of the system—depicted in FIG. **8A**-*comprises* entities that are not specific to one of the tiers of the QuestObjects system. One of the most important entities shown in FIG. **8A** is QoString, the QuestObjects String. QoString models the strings that the QuestObjects System handles. A QoString has at least a value, which is the sequence of (Unicode) characters itself. To guarantee a minimum performance level, i.e. one in which the communication takes as little time as possible, this value has a limited length (e.g. of 256 characters). Furthermore, a QoString may have a key and metadata. The key (if any is present) is the identifier (i.e. the primary key) of the QuestObjects String in the database from which it originates. This key can be used to retrieve data from the database that is related to the QuestObjects String. Metadata of a QoString can be any additional data that is provided with the QoString's value. Metadata of a QoString is XML formatted and has a limited length (e.g. 2048 bytes), in order to ensure that QoStrings can be exchanged between the tiers of the QuestObjects System without compromising efficiency. If the QoString originates from a Content Channel, it may also have a fetchTime, namely the timestamp of when the QoString was retrieved from the underlying content provider. It also may have an expirationTime indicating how long

the data in the QoString is to be considered valid. Optionally a QoString can have a type, which is a reference to a QoType object. (Note that for maximum efficiency the types are not actually stored in the QoStrings, because it is very likely that many QoStrings in a QoResultSet have the same type. Storing the types in the strings would unnecessarily increase network traffic.)

The QoType object models the concept of a string's type. It has a string typeString that contains the description of the type and an indicator typeIndicator that defines the meaning of the description (typeString). Examples of string types are: the DTD or Schema of the string's value in these cases in which it is XML formatted (or, alternatively, the URL of the DTD or Schema), the number formatter in the case it is a number, and the date (and/or time) formatter in the case it is a date (and/or time). Table 1 shows an example of the use of types, especially type indicators.

TABLE 1

Value of typeIndicator	Meaning of typeString
0	typeString contains the name of the type
64	typeString contains a string formatter
65	typeString contains a number formatter
66	typeString contains a date formatter
128	typeString contains a DTD
129	typeString contains a Schema
160	typeString contains the URL of a DTD
161	typeString contains the URL of a Schema
255	custom type; typeString is the type's name

In the example shown in Table 1, bit **7** of the typeIndicator is on if typeString is XML related, bit **6** is on if typeString is some formatter, and bit **5** is on when typeString is a URL. This name must follow the same naming scheme as Java packages: They must use the Internet domain name of the one who defined the type, with its elements reversed. For example, custom types defined by MasterObjects would begin with "com.masterobjects.".

The QoQuery entity models the specification of a QuestObjects Query. It includes a queryString that contains the value the Content Channel is queried for (which is named queryString in the figure). In addition to the queryString, QoQuery has a property 'qualifier' that can hold any other attributes of the query. The format and meaning of the qualifier's contents is defined by the Content Channel that executes the query. Furthermore, it can be specified which row numbers of the total result set must be returned using the property 'rownums'. The property 'requestedTypes' can optionally hold a list of QoTypes, limiting the types of the strings that will result from the query. The 'timeout' property can be used to specify a maximum amount of time execution of the query may take.

Queries may include a type (QoQuerytype). Query types are similar to QoType (i.e. String Types), and can be used by QuestObjects Clients to find all QuestObjects Services that support a certain kind of Query.

The result of a query is represented by the QoResultSet entity. QuestObjects Result Sets are collections of QuestObjects Strings that are sent from a QuestObjects Server to a QuestObjects Client in response to a query. QoResultSets are created and filled by a QuestObjects Service (to which QoResultSet has a reference named 'service'), based on the QoQuery to which the QoResultSet has a reference. Actual results are stored as an array of QoStrings in the 'strings' property. Elements of the QuestObjects Result Set (i.e. QoStrings) may be selected, as indicated by the 'selected' property that is a list of indices in the strings array of selected strings. Also, one of the QoStrings may be marked as current as indicated by the 'current' property. (When a QoString is marked as current it means that all operations are performed on that QoString, unless another one is explicitly specified.) QuestObjects Result Sets include an attribute 'ordered' that indicates whether the QoStrings in the QoResultSet are ordered. Sometimes, especially when a QuestObjects Result Set is narrowed down by a new Query, the fact that the QoResultSet is ordered may mean that the QuestObjects Client does not need to actually execute a new Query; instead, it can filter the previous QuestObjects Result Set itself according to the new queryString.

As further described below, Server Questers may have a QuestObjects Result Set, of which only a part is sent to the QuestObjects Client. At all times, the 'rownums' property of QoResultSet indicates the row numbers of QoStrings that are actually present in the QoResultSet. The rownums property may have different values for corresponding QoResultSets on the QuestObjects Server and the QuestObjects Client. The same holds for the 'strings' property. The 'complete' property is the percentage of the QoStrings in the server-side QoResultSet that is present in the corresponding client-side QoResultSet as well. The property 'totalNumberOfStrings' indicates the total number of QoStrings in the QoResultSet, whether actually present or not. For server-side QoResultSets this number is always equal to the length of the 'strings' array, but for client-side QoResultSets the number may be smaller.

Finally, result sets include an identifier 'resultSetId'. Every time a Client Quester uses the protocol of the present invention to send something to the Server Quester that may result in a new QuestObjects Result Set, it includes a request identifier. This identifier is then copied in the resultSetId when the QuestObjects Result Set is sent to the Client Quester. In this way Client Questers know which request the QuestObjects Result Set belongs to. (This is important because the system is asynchronous and on occasions it may occur that a newer QuestObjects Result Set is sent to the client before an older one. The request identifier and QuestObjects Result Set identifier allow the Client Quester to detect and handle this.)

The core entity in the figure is QoQuester. QoQuester is the superclass of both QoClientQuester (part of the client and thus depicted in FIG. 8B) and QoServerQuester (depicted in FIG. 8C). The QoQuester entity models the Quester concept. Its primary task is to maintain an input buffer, to make sure that QuestObjects Queries are executed and to store and provide access to the QuestObjects Result Sets returned by QuestObjects Services in reply to QuestObjects Queries. At all times, a QoQuester holds one QoResultSet that contains the results of the latest QuestObjects Query. (Note that a QoQuester may hold other QoResultsSets as well, for example for optimization purposes.) Client Questers and Server Questers exist in a one-to-one relationship with each other: for every Client Quester there is exactly one corresponding Server Quester, and vice versa. All properties listed in QoQuester are present and equal, both in the Client Quester and in the corresponding Server Quester. An important exception is the resultSet property. In the Server Quester, this is always the entire QuestObjects Result Set of the latest Query. However, in order to minimize network traffic the Server Quester is intelligent about the portion it actually sends to the Client Quester. Questers include a property 'minimumBatchTime' that indicates the minimum amount of time that should pass before the Server Quester sends results to the Client Quester. This allows the Server Quester to accumulate results and send them as a single action instead of as a separate action for each result. There are two situations in which the Server Quester may ignore this minimum batch time:

(a) when the result set is complete before the minimum batch time has passed, and

(b) when the number of accumulated results exceeds the number indicated by the 'resultSetBatchSize' property before the minimum batch time has passed.

If, for whatever reason, the Server Quester postpones sending the accumulated results to the Client Quester, the (optional) 'maximumBatchTime' property indicates how long it may postpone the sending. Even if no results are available yet, when the maximumBatchTime passes, the Server Quester must notify the Client Quester thereof.

Results are sent to the Client Quester in batches, the size of which is indicated by the 'resultSetBatchSize' property. Occasionally, the Server Quester may deviate from this batch size, notably when the number of results that is not present on the client is smaller than the batch size or when the maximumBatchTime has passed. This concept can be taken even further, for example when the batch size is 10 results and the Server Quester has 11 results, the Server Quester may send them all, even though it exceeds the batch size, because sending one extra result with the other 10 is probably more efficient than sending a single result in a separate batch at a later point. The Server Quester can use the 'clientMaximumLatency' to make such decisions; it indicates the maximum expected amount of time that elapses between sending a message and receiving its response. The higher this value, the more likely it is that sending the eleventh result with the other ten is more efficient.

Questers include an input buffer. The content of the input buffer is what the QuestObjects Service will be queried for. In the Client Quester, the input buffer is controlled by the application that uses the QuestObjects system. For example, an application with a graphical user interface may update the input buffer according to key presses in one of its input fields. The Client Quester keeps the input buffer of its corresponding Server Quester up to date using the protocol of the present invention.

Properties 'highestReceivedResultSetId' and 'latestRequestId' are used to detect when QuestObjects Result Sets are received out of order. As with the 'resultSetId' property of the QoResultSet, every QuestObjects Result Set includes an identifier. The 'highestReceivedResultSetId' property stores the highest of all received QuestObjects Result Set identifiers. If a Client Quester only needs the latest results, it can simply discard received QuestObjects Result Sets that have a lower identifier than 'highestReceivedResultSetId'. The 'latestRequestId' is the identifier of the latest request. The QuestObjects Result Set with an identifier that matches 'latestRequestId' holds the results of the latest request.

The remaining properties of QoQuester store the QuestObjects Service the Quester uses ('service'), the optional qualifier that Queries to this QuestObjects Service need ('qualifier'), the types the Quester can handle ('types'), whether an application proxy is needed, and the optional function of the Quester in the application ('applicationFunction', used by the application proxy mechanism to determine how the value of the Quester is to be passed to the application/web server). In addition, if the update interval property 'autoUpdateInterval' is set to a non-zero value, the Server Quester will automatically repeat the last Query with that interval. This is useful for QuestObjects Services that are not capable of pushing results themselves. A mechanism is required to allow any other entity to be notified of changes in the Quester. There are many ways this can be done. As an example in the embodiment shown in FIGS. 8A-8D an event mechanism is included that involves event listeners and event handlers, very similar to the Java2 event mechanism. An entity that wants to be notified of changes must implement the QoQuesterChangeListener interface and then be added to the Quester's 'changeListeners' property (using the method 'addQuesterChangeListener'). When the Quester changes, it will call the 'questerChanged' method of all registered QoQuesterChangeListeners with a QoQuesterChangeEvent as a parameter. The QoQuesterChangeEvent holds a description of the changes of the Quester; it has a reference to the Quester that raised the event and an event type. In FIG. **8** three event types are displayed (INPUT_BUFFER_CHANGED indicates that the Quester's input buffer has changed, RESULT_SET_CURRENT_CHANGED indicates that the current item of the Quester's Result Set has changed, and RESULT_SET_SELECTED_CHANGED indicates that the list of selected results in the Quester's Result Set has changed). More event types can be added as desired.

Another important entity in FIG. **8**A is QoController. QoController is the entity that implements the protocol of the present invention. In addition, it knows how to buffer usage statistics and also handles the caching of result sets. QoController includes two subclasses (QoClientController and QoServerController), depicted in FIG. **8***b* and FIG. **8***c*, respectively. Buffering of usage statistics is an optimization that eliminates the need of exchanging usage data between the layers of the system every time a result is used. Instead, the QuestObjects Controller buffers that data and flushes the buffer when the statisticsBufferFlushTime has passed. Caching is an optimization as well. Caching is done by the QoResultsCache entry, to which the QuestObjects Controller has a reference. The QoResultsCache has a list of cached entries ('resultsCacheEntries'). The entry of the cache is modeled as QoResultsCacheEntry, an entity that has a list of QuestObjects Result Sets for combinations of query strings and qualifiers (as defined in QoQuery).

The last entity in FIG. **8**A is QoQueryValidator. QoQueryValidator is an abstract class that defines the method 'is Valid'. This method has a query string as a parameter and returns either 'true' or 'false'. QuestObjects Services may declare and publish a QoQueryValidator. By doing so, they allow the QuestObjects Server to verify the validity of a query string without actually having to send it to the QuestObjects Service, thus eliminating network traffic for invalid query strings.

FIG. **8**B displays the minimal entities every QuestObjects Client must have. Every client of the QuestObjects System at least has a Client Controller QoClientController. QoClientController is a subclass of QoController that implements the client side of the protocol of the invention. Applications using the QuestObjects System do so through Client Questers, modeled as QoClientQuester. QoClientQuester is the subclass of QoQuester that implements client-specific Quester functionality. The figure contains the entity 'ActiveComponent'. It represents some entity that uses the QuestObjects System through one or more Client Questers.

FIG. **8**C shows the server part of the embodiment of the present invention, and includes the QoServerQontroller, one of the subclasses of QoController. QoServerController implements the server-side part of the protocol of the present invention. In addition, it maintains a list of sessions running on the server, and it has references to a Persistent Quester Store, an optional Service Directory and a list of optional Application Host Synchronizers. For security reasons, one implementation of the QuestObjects System may require that only certified clients can connect to the system. A boolean 'requiresCertification' indicates this.

The QuestObjects System is session-based. This means that clients that use the system are assigned to a session, modeled by the QoSession entity. Every session has a unique identifier, the 'sessionId'. The QoSession entity maintains a list of Server Questers that are active in the session (stored in the 'serverQuesters' property). Furthermore, it has a reference to the Server Controller through which a QuestObjects Client is using the session.

QoServerQuester is the server-side subclass of QoQuester. It includes a reference to the session it is being used in (the 'session' property). Furthermore, when the QuestObjects Service that the Quester uses has a Query Validator, QoServerQuester has (a reference to) a copy of that Query Validator, so that query strings can be validated before they are actually sent to the QuestObjects Service. The QoPersistentQuesterStore is an entity that is able to store a user's session and to restore it at some other time, even when the session would normally have expired or even when the same user is connecting from a different client machine. To this end, QoServerQuester has two methods 'store' and 'restore'. The first, 'store', returns a QoStoredQuester, which is a (persistent) placeholder of the Server Quester that contains all relevant data of that Server Quester. The second, 'restore', needs a QoStoredQuester as an argument. The two are each other's inverse, which means calling 'store' on a QoServerQuester and then calling 'restore' on the result creates a new QoServerQuester that is an exact copy of the original QoServerQuester.

QoServiceDirectory acts as a Yellow Pages or directory of QuestObjects Services. For each QuestObjects Service it stores the name and address, as well as the address of the QuestObjects Server through which the Service can be accessed. Furthermore, Services' profiles are additionally stored to allow clients to find all QuestObjects Services satisfying desired criteria.

Finally, QoAppHostSynchronizer is the AppHost Synchronizer. QoAppHostSynchronizer has its address as a property ('appHostAddress').

FIG. **8**D depicts the service part of the embodiment of the present invention. Content is disclosed through Content Channels (the QoContentChannel entity). Content Channels use Content Access Modules (QoContentAccessModule) to obtain their data in a standardized way, so only the Content Access Module knows how to communicate with the underlying data source. Content Channels are organized in Syndicators (the QoSyndicator entity), and each syndicator includes a list of Content Channels. Each Quester in the QuestObjects System uses a specific Content Channel of a specific Syndicator. This is called a QuestObjects Service, namely one of the Content Channels of a Syndicator. The property 'subscriptionRequired' indicates whether the user needs to be a registered user to be allowed to use the Service. If it is false, only users listed in 'users' may use the Service. Users can be subscribed to QuestObjects Services, which is modeled by the QoSubscription entity. Statistics are kept per Content Channel using the QoUsageStatisticsStore entity. Content Engines optionally have a Query Validator that the QuestObjects Server may use to validate Query Strings before sending them off to the QuestObjects Service. In addition, Content Channels have a profile that consists of a Content Channel's description, a list of types (QoType) of QuestObjects Strings the Content Channel can provide, an optional list of DTDs of that metadata of QuestObjects Strings from the Channel conforms to, and an optional list of Query Types the Content Channel accepts.

QuestObjects Servers communicate with QuestObjects Services through the QoServiceSession. The QoServiceSession has a static reference to the QuestObjects Service it belongs to, as well as a static array of IP addresses of QuestObjects Servers that are allowed to connect to the QuestObjects Service. In some versions of the QoServiceSession the array of IP addresses can be replaced by a list of addresses and netmasks, or by IP address ranges. Every instance of QoServiceSession has the IP address of the server that is using the session ('serverAddress'), a connectionTimeout indicating the maximum period of idle time before the Service Session is automatically ended, and a serviceSessionId that can be used to refer to the Service Session.

As described above, a QuestObjects Service is one of the Content Channels of a Syndicator, so QoService has a reference to both ('syndicator' and 'contentChannel'). The property 'listable' indicates whether the Service may be listed in a Service Directory (server::QoServiceDirectory). If not, the Service can only be used if the application writer (i.e. the programmer using the QuestObjects to develop an application) knows that it exists and where it is available. The property 'name' is the Service's name, used in the Service Directory amongst others. This name must use the same naming scheme as the names of custom types. The boolean 'subscriptionRequired' indicates whether users must be subscribed (modeled by QoSubscription) to the Service in order to be allowed to use it. If the Content Engine of this Service's Content Channel requires login, 'contentEngineLoginName' and 'contentEngineLoginPassword' are the name and password with which is logged in. Finally, 'pricingInfo' contains information about the costs involved in using the Service. It is formatted as XML, conforming to a well-defined structure (i.e. DTD or Schema).

A Content Channel has a name (the 'name' property) and a profile (QoContentChannelProfile). The profile provides information about the Content Channel, namely about the Query Types it accepts ('queryTypes'), the types of the Strings it can provide ('types'), and the DTDs that QuestObjects Strings' metadata conforms to. In addition, it has a textual 'description' of the content the Content Channel discloses.

Content Channels also have properties that define the criteria Query Strings have to satisfy. The property 'queryStringMinLength' defined the minimum length a valid query has. Alternatively or additionally, 'queryStringRegularExpressions' may contain a list of regular expression describing valid Query Strings (meaning that Query Strings have to match at least one of the regular expressions). The property 'queryStringFilters' may hold a list of regular expressions and replacement strings that can transform Query Strings in a well-defined manner (for example the way the standard Unix utility 'sed' does it). Instead of using these three properties, Content Channels may define a QoQueryValidator (described above in FIG. 8A). If there is a Query Validator, 'queryStringMinLength', 'queryStringRegularExpressions', and 'queryStringFilters' are ignored.

As described above, Syndicators may have a list of users. Users (QoUser) have a name and a password, as well as a list of subscriptions (QoSubscription). QoSubscription models a user's subscription to a Service (the 'service' property). The properties 'startDate' and 'expirationDate' define the time frame during which the subscription is valid. Outside that time frame the user will be denied access through the subscription. The maximum number of queries the user may run in the Service is stored in the 'queryLimit' attribute. The 'queryLimitReset' defines when the query counter is reset. For example, if queryLimit is 10 and queryLimitReset is 7

days, the user may run **10** queries per week. (If queryLimit equals zero the number of queries is unlimited and queryLimitReset is ignored.) The property 'resultLimit' stores the maximum number of results the user may receive from the subscription. Similar to 'queryLimitReset', 'resultLimitReset' defines how often the result counter is reset. If 'resultLimit' equals zero the number of results is unlimited and 'resultLimitReset' is ignored. The property 'pushAllowed' indicates whether the user may use the Service in pushing mode. If so, 'pushIntervalLimit' indicates the minimum amount of time that has to pass between two pushes. A 'historyAllowed' variable indicates whether a history is kept of the use of the subscription; if so, 'historyLimit' indicates the maximum size of the history. If the maximum size is exceeded, the oldest history data is deleted so that the size of the history is below the maximum size again. If 'historyLimit' equals zero, the size of the history is unlimited. Finally, a 'usageAnonymous' variable indicates that the QoUsageRecords that are generated for this subscription must not contain user information (this is necessary because of privacy issues).

If 'keepServiceStatistics' is true, then the QoUsageStatisticsStore can store three kinds of statistics:

statistics about Strings that have been displayed on the client; the 'keepClientDisplayedStatistics' indicates whether this kind of statistics are kept.

statistics about Strings that have actually been selected on the client; the 'keepClientSelectedStatistics' indicates whether this kind of statistics are kept.

statistics about Strings that have a used on the client; the 'keepClientUsedStatistics' indicates whether this kind of statistics are kept.

The Client Quester determines the exact meaning of the three kinds of statistics. In the case of web applications, a string is generally considered displayed when the Client Quester accesses it in its QuestObjects Result Set. It is considered selected when a new Query is executed with the String as Query String. It is considered used when the form on which the Client Quester is active is submitted with that String. The actual data is stored as a list of QoUsageRecords in the property 'records'.

A QoUsageRecord holds usage information about a QuestObjects String or a number of QuestObjects Strings. If, in one Service Session, a Quester gets the same Result Set more than once (consecutively), the usage data of each of the Strings in the Result Set is grouped in one QoUsageRecord. However, if 'stringKey', 'stringValue', 'rowInResultSet', or 'totalRowsInResultSet' changes, a new QoUsageRecord must be used from that point on. The properties of QoUsageRecord mean the following:

stringKey: if available, this is the unique key of the QuestObjects String as provided by the Content Access Module.

stringValue: the value of the QuestObjects String.

rowInResultSet: the row of the QuestObjects String in its QuestObjects Result Set.

totalRowsInResultSet: the number of rows the QuestObjects String's Result Set had.

dateReturnFirst: the timestamp of the first time the QuestObjects String was returned by the Content Channel.

dateReturnLast: if the QoUsageRecord represents a group of usage events, this is the timestamp of the last event.

clientDisplayed: indicates whether the QuestObjects Client that received the QuestObjects String considers it to be displayed.

clientSelected: indicates whether the QuestObjects Client that received the QuestObjects String considers it to be selected.

clientUsed: indicates whether the QuestObjects Client that received the QuestObjects String considers it to be used.

applicationName: the name of the application to which the Quester that received the QuestObjects String belongs.

appliationFunction: the function (if available) of the Quester that received the QuestObjects String.

activeComponentId: the identifier of the Active Component that received the QuestObjects String.

user: the identifier of the user that saw/selected/used the String. If the user's subscription has 'false' as value of 'usageAnonymous', then this property is empty.

Queries are executed by QoQueryExecutors. A Query Executor has a reference to the Service Session in which the Query is executed, it has a reference to the Query itself, and it also has a reference to the Server Quester that has the Query executed. This reference may be a remote object when Corba is being used, for example. If some proprietary protocol is used, it may just be the unique identifier of the Server Quester.

FIG. **9** shows a method for using the present invention in systems that have limited technical capabilities on the Client side, such as, for example, web browsers with embedded Java applets. If developers of client systems have not integrated Client components of the present invention into their client software, then Client components needed for the present invention must be present as Plug-Ins, DLL's, or an equivalent device, or they must be downloaded to the client computer as applets. These applets can be written in the Java language, when they are needed. For security reasons, such Client systems including web browsers usually do not allow 'foreign' software (i.e. software that is not an integral part of the web browser) to influence or change data entered by the user before it is sent to the application server (in this case the web server). Without an additional infrastructure on the server side, the present invention could not easily be used to enter data by users of systems with such limited technical capabilities on the client, because data entered and selected using the present invention would not be communicated to the existing application/web server. However, the modified invention and method described in FIG. **9**, referred to as an Application Proxy, offers a solution.

Although the system depicted in FIG. **9** can be used to support clients in practically any server-based application server, and particularly in the case of a web server hosting an application used by end users to enter data that is partially retrieved using the present invention, the system is not limited to the web. The system provides an ideal solution for current web-based applications that consist of web browsers **903** on the client side and web host computers **901** with web server software **917** on the server side. To allow the web server **917** to access data selected using the present invention, this system provides a link between the web server and the QuestObjects Server **902**. In this case, QuestObjects Server acts as a data-entry proxy between the existing client system (web browser) and the existing web server. Data entered by the client is submitted to the QuestObjects Adaptor instead of to the web server. The QuestObjects Adaptor then fills in the values of the Questers and passes the data to the web server. An Application Proxy is not required if the QuestObjects Client components can directly insert data into the client entry form on the web browser, as is the case on certain platforms that allow integration between Java applets or other components and JavaScript in the web browser.

In FIG. **9**, the web server runs on a host computer **901** typically associated with a fixed IP address or an Internet host

name. The web server is accessed by any number of clients using web browsers **903**. To allow users to enter data and send data to the server, web pages make use of HTML forms **904**. To use the present invention, user interface elements such as entry fields in these HTML forms are associated with Questers **905** in the form of browser Plug-Ins or Java Applets. Through a QuestObjects Controller **906** those Questers allow the user to access one or more QuestObjects Services hosted by a QuestObjects Server **902** using the protocol of the present invention **907**. The Server Controller **908** forwards user actions generated in the Client Questers **905** to their corresponding Server Questers **909** that thus are always aware of data selected in the Client. When a Server Quester is first activated, it checks whether it is being used by a client system that requires the use of an Application Proxy. If the answer is yes, then the Quester creates a corresponding AppHost Synchronizer **911** that contacts the QuestObjects Adaptor **914** on the host computer **901** using a standardized protocol **915**. The QuestObjects Adaptor then knows which QuestObjects Server to contact to retrieve QuestObjects data **915** after the user submits form data **912** to the application host using the existing application protocol **913**, such as HTTP POST or HTTP GET. The QuestObjects Adaptor then replaces the appropriate form field data with the strings selected in the Server Questers **909** before forwarding this form data, now including data selected using the present invention, to the web server **917**.

Design Implementation

The preceding detailed description illustrates software objects and methods of a system implementing the present invention. By providing a simple and standardized interface between Client components and any number of Content Engines that accept string-based queries, the present invention gives content publishers, web publishers and software developers an attractive way to offer unprecedented interactive, speedy, up-to-date and controlled access to content without the need to write an access mechanism for each content source.

In addition to acting as a standardized gateway to any content engine, the present invention can intelligently cache query results, distribute Services over a network of Servers, validate user and other client input, authorize user access and authenticate client software components as needed. These and other optional services are provided by the present invention without requiring additional work on the part of software developers or content publishers. Publishers can also keep track of usage statistics, on a per-user basis as required allowing flexible billing of content access. Content Access Modules allow software developers and vendors of Content Engines such as database vendors and search engine vendors to create simplified ways for developers and implementers of such content engines to disclose information through the present invention.

End users of the present invention experience an unprecedented level of user-friendliness accessing information that is guaranteed to be up-to-date while being efficiently cached for speedy access as the number of simultaneous users grows.

The present invention can be implemented on any client and server system using any combination of operating systems and programming languages that support asynchronous network connections and preferably but not necessarily preemptive multitasking and multithreading. The interface of the present invention as it appears to the outside world (i.e. programmers and developers who provide access to end users and programmers who provide Content Access Modules to Content Engines used by content publishers) is independent of both the operating systems and the programming languages used. Adapters can be built allowing the tiers of the system to cooperate even if they use a different operating system or a different programming language. The protocol of the present invention can be implemented on top of networking standards such as TCP/IP. It can also take advantage of inter-object communication standards such as CORBA and DCOM. The object model of the present invention can be mapped to most other programming languages, including Java, C++, Objective C and Pascal.

Third-party vendors of software development and database management tools can create components that encapsulate the present invention so that users of those tools can access its functionality without any knowledge of the underlying protocols and server-side solutions. For example, a 4GL tool vendor can add an 'auto-complete field' to the toolbox of the development environment allowing developers to simply drop a Questlet into their application. In order to function correctly, the auto-complete field would only need a reference to the QuestObjects Server and one or more QuestObjects Services, but it would not require any additional programming.

Examples of Applications in which the invention may be used include: Access system for database fields (for lookup and auto-complete services); Enterprise thesauri system; Enterprise search and retrieval systems; Enterprise reference works; Enterprise address books; Control systems for sending sensor readings to a server that responds with appropriate instructions or actions to be taken; Client access to dictionary, thesaurus, encyclopedia and reference works; Access to commercial products database; Literary quotes library; Real-time stock quote provision; Access to real-time news service; Access to Internet advertisements; Access to complex functions (bank check, credit card validation, etc); Access to language translation engines; Access to classification schemes (eg, Library of Congress Subject Headings); Access to lookup lists such as cities or countries in an order form; Personal address books; and, Personal auto-complete histories.

The foregoing description of preferred embodiments of the present invention has been provided for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Obviously, many modifications and variations will be apparent to the practitioner skilled in the art. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, thereby enabling others skilled in the art to understand the invention for various embodiments and with various modifications that are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalence.

What is claimed is:

1. A system comprising:

a server system, including one or more computers, which is configured to receive query messages from a client object, the server system asynchronously receiving and responding to the query messages from the client object over a network;

the client object that, while a user is providing input comprising a lengthening string of characters, sends query messages to the server system;

whereby the query messages represent the lengthening string as additional characters are being input by the user; and

wherein the server system, while receiving said query messages, uses the input to query data available to the server system and send return messages to the client object containing results in response to the input; and

wherein, upon receiving a return message of the return messages from the server system, the client object tests the usability of the results in the return message by checking that the return message corresponds to the latest query, and if usability is established, the client object displays or returns at least some result data to the user.

2. The system of claim **1**, wherein, upon testing the usability of the server system results, at least some result data is displayed as an auto-completion inside of an input field.

3. The system of claim **1**, whereby the lengthening string is entered into an input field, and wherein upon testing the usability of the server system results, at least some result data is displayed in a separate area that is associated with the input field or that pops up near said input field.

4. The system of claim **1**, whereby the lengthening string is entered into an input field, and wherein one or more symbols displayed inside of the input field indicate(s) to the user one or more of whether or not said system is present, whether the system is available for use, the current state of the system, whether a query has been sent to the server system, whether more results are available, whether a previous result is available, whether a next result is available, or whether the current result is the only available match.

5. The system of claim **1**, wherein the server system sends return messages to the client object containing results both in response to the input and associated with a string contained elsewhere on the same client object to which the input has a predefined dependency.

6. The system of claim **1**, wherein the server system retrieves the results from one or more of a database, a search and retrieval system, a thesaurus, a reference work, an address book, a control system, a dictionary, an encyclopedia, a products database, a quotes library, a stock quote system, a news service, internet advertisements, a catalog, a complex function, a translation engine, a classification scheme, a lookup list, an auto-complete history, an algorithm, a directory, a search engine, a database retrieval engine, or a cache.

7. The system of claim **1**, wherein the server system caches query results and subsequently determines results by looking up the query in said cache so that it can avoid performing a query for the same input on a data source or looking up said query in a second cache.

8. The system of claim **1**, wherein the client object transmits an associated query message to the server system upon each detected change to the input.

9. The system of claim **1**, wherein the client object accumulates input before transmitting an associated query message to the server system.

10. The system of claim **1**, wherein the client object combines the input string with additional information, whereby said additional information includes one or more of an indication of whether or not results should be sorted, whether results should be in response to both the user input and a qualifier, how many results should be returned, or which selection of results should be returned.

11. The system of claim **10**, whereby said qualifier identifies a user to the server system whereby the server system returns messages containing results in response to said user.

12. The system of claim **1**, wherein the results returned by the server system include suggestions for the user input; and

wherein these suggestions change dynamically while the user is providing input.

13. The system of claim **1**, wherein selections of results returned by the server system are related to the user input through predefined relationships; and wherein an indicator of the corresponding relationship is displayed or returned alongside each of said result selections.

14. The system of claim **13**, wherein said relationships are organized according to a dictionary or thesaurus system that includes one or more of broader term relationships, narrower term relationships, related term relationships, synonym relationships, used-for term relationships, meaning relationships, or uses relationships.

15. The system of claim **1**, wherein results returned by the server system comprise result sets consisting of zero or more string values.

16. The system of claim **1**, wherein results returned by the server system comprise a set of zero or more results;

wherein each result consists of one or more of a string, key, fetch time, expiration time, metadata, logical link to other data sources, or a Uniform Resource Identifier.

17. The system of claim **1**, wherein the client object determines the usability of each server system response by comparing an original input to a then-current input; and

wherein the client object deems the results usable if they match.

18. The system of claim **1**, wherein the query message sent to the server system includes a request identification that is included by the server system in the corresponding server response message.

19. The system of claim **18**, wherein the usability of a server system response is determined by the client object by matching the request identification received in the server response message against a request identification on the client.

20. The system of claim **1**, wherein the client object caches results received from the server system and reuses said cached results when Previously Presented queries match queries contained in the cache or if cached query results can be filtered to match the Previously Presented queries, instead of sending messages representing those Previously Presented queries to the server system.

21. The system of claim **1**, wherein one or more filters are used to validate or transform the input string using a type, pattern, or minimum length; and

wherein no query is performed if the input string is found not to conform to or does not transform using said type, pattern, or minimum length.

22. The system of claim **1**, wherein the server system is capable of returning results from multiple data sources;

wherein the client object selects which of the available data sources at the server system is to be queried; and

wherein the system selects one or more data sources based on a name associated with each data source, on types of queries accepted by each data source, or on string types that can be returned by each data source.

23. The system of claim **1**, wherein the input on the client object represents speech and is generated by a sound conversion engine.

24. The system of claim **1**, wherein return messages include suggestions and related data relevant to the suggestions, and wherein the related data is displayed in a user selectable manner; wherein a selection of the related data displayed to the user causes additional data to be obtained from the server system and be displayed.

25. The system of claim **1**, wherein the client object is run by a web browser.

26. The system of claim **1**, wherein the client object is run on a mobile device.

27. The system of claim **1**, wherein the client object tests the usability of the results in the return message by matching an ID for the user query.

28. The system of claim **27**, wherein the client object tests the usability of the results in the return message by matching an ID included in one of the query messages sent to the server system and returned as part of the return message.

29. The system of claim **1** wherein the client object uses a pre-defined query and automatically transmits a corresponding message to the server as the client object is first run, and wherein user input is not required before server responses are sent to the client object.

30. The system of claim **1**, wherein the server system automatically sends messages containing Previously Presented results to the client object as updated data in response to a previous query becomes available.

31. The system of claim **1**, wherein the client object automatically repeats a query to retrieve updated information from the server system.

32. A system including at least one computer comprising:

a server system using a communication protocol that enables asynchronous communication between the server system and a client object; and

wherein the client object that, while a user is providing input comprising a lengthening string of characters, sends query messages to the server system;

whereby the query messages represent the lengthening string as additional characters are being input by the user; and

wherein the server system, while receiving said query messages, uses the input to query data available to the server system and send return messages to the client object containing results in response to the input

wherein upon receiving corresponding return messages from the server system, the client object tests the usability of each return message by checking that the return message corresponds to the latest query, and if usability is established, provides feedback to the user based on the contents of the return message.

33. The system of claim **32**, wherein the client object is run using a web browser.

34. The system of claim **32**, wherein the client object is run on a mobile device.

35. A system comprising:

a client object adapted to receive input comprising a lengthening string of characters from a user, the client object asynchronously sending multiple query messages corresponding to multiple versions of said input to a server system while a user modifies the input, comprising a lengthening string of characters, the client object receiving return messages with results in response to the multiple versions of the input;

whereby the query messages represent the lengthening string as additional characters are being input by the user; and

wherein the server system, while receiving said query messages, uses the input to query data available to the server system and send return messages to the client object containing results in response to the input

wherein upon receiving one of the return messages from the server system, the client object checks the usability of the results of the one of the return messages using a more recent version of the input to determine whether to display at least some of the results of the one of the return messages to the user.

36. A system comprising:

a server system, including one or more computers, which is configured to receive query messages from a client object, the server system asynchronously receiving and responding to the query messages from the client object over a network;

wherein the client object, while a software process is providing input comprising a lengthening string of characters, sends query messages representing said input, to the server system;

whereby the query messages represent the lengthening string as additional characters are being input by the software process;

wherein the server system, while receiving said query messages, uses the input to query data available to the server object and send return messages to the client object containing results in response to the input; and

wherein, upon receiving a return message of the return messages from the server system, the client object tests the usability of the results in the return message by comparing the return message to the then-current input or matching it with a request identification maintained on the client object, and if usability is established, the results are returned to the software process.

37. A system comprising:

a server system, including one or more computers, which is configured to receive query messages from a client object, the server system asynchronously receiving and responding to the query messages from the client object over a network;

the client object that, while a user is providing input comprising a lengthening string of characters, sends query messages representing said input to the server system;

whereby the query messages represent the lengthening string as additional characters are being input by the user;

wherein the server system, while receiving said query messages, uses the input to query data available to the server system and send return messages to the client object containing results in response to the input; and

wherein, upon receiving a return message of the return messages from the server object, the client object tests the usability of the results in the return message by matching an ID associated with the input sent to the server system with an ID maintained in the client object, and if usability is established, the client object displays or returns at least some of the result data to the user.

* * * * *

EXHIBIT B

FLIESLER MEYER LLP

INTELLECTUAL PROPERTY LAW

650 CALIFORNIA STREET • FOURTEENTH FLOOR
SAN FRANCISCO • CALIFORNIA 94108
TELEPHONE 415.362.3800
FACSIMILE 415.362.2928
INTERNET WWW.FDML.COM

MARTIN C. FLIESLER
mcf@fdml.com

June 27, 2008

Kent Walker
VP and General Counsel
Google Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043

Marissa Mayer
VP of Search Products & User Experience
Google Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043

Re: **MasterObjects**

Dear Mr. Walker and Ms. Mayer:

This letter is with respect to the technology identified by Google Inc. ("Google") on its website as Google Suggest, Google Desktop and various dynamic information retrieval functions integrated into Google's applications and prior related technology developed by MasterObjects, Inc., San Francisco, California ("MasterObjects") identified as QuestFields. MasterObjects, through its research and development has, since the late 1990's, invested in and developed its body of QuestFields technology. MasterObjects is acquiring a U.S. and foreign patent portfolio related to this QuestFields technology. MasterObjects owns pending published patent applications and pending unpublished patent applications, U.S. and foreign, related to its work. MasterObjects also has a significant body of software technology, including source code, that is readily installable for on-line use by its customers and potential customers.

Google's business strategy is all about focus on information that the user needs, about simplicity and instant user gratification. QuestFields technology provides a SOA-based solution that helps people find and retrieve information interactively, be it through traditional web pages, in mashups, or on mobile devices. QuestFields technology centers around a single challenge: Enabling improved interaction with content sources and providing immediate validation in order to further reduce the time it takes for users to find what they need.

You may learn more about MasterObjects by accessing its website at:

http://www.masterobjects.com/

We represent MasterObjects and have represented MasterObjects since its inception in the late 1990's. The purpose of this letter is to ask if Google would be interested in exploring a world-wide licensing arrangement with MasterObjects as licensor and Google as licensee, and having discussions with MasterObjects about its software development work and know-how. Alternatively, MasterObjects, as part of an exit strategy, is willing to entertain a merger or acquisition offer.

FLIESLER MEYER LLP

We have therefore attached for your convenience copies of the following, publicly available MasterObjects-owned U.S. patent related documents (and their published foreign counterparts):

Tab A:	Published Patent Application No. US2003-0041147 (European Publication No. 1425677)
Tab B:	Published Patent Application No. US2006-0075120 (European Publication No. 1653376)

We are contacting other entities on behalf of MasterObjects that might be interested in a similar manner in this opportunity.

There is no date by which we request your response, but if you are interested in pursuing matters, we do look forward to hearing from you.

Very truly yours,

[illegible]

Martin C. Fliesler

MCF/mmc
Enclosures
M:\mcf\wp\mobj\License-Google.doc

A

US 20030041147A1

(19) United States
(12) Patent Application Publication (10) Pub. No.: US 2003/0041147 A1
van den Oord et al. (43) Pub. Date: Feb. 27, 2003

(54) SYSTEM AND METHOD FOR ASYNCHRONOUS CLIENT SERVER SESSION COMMUNICATION

(76) Inventors: Stefan M. van den Oord, Weesp (NL); Mark H. Smit, Maarssen (NL)

Correspondence Address:
FLIESLER DUBB MEYER & LOVEJOY, LLP
FOUR EMBARCADERO CENTER
SUITE 400
SAN FRANCISCO, CA 94111 (US)

(21) Appl. No.: 09/933,493

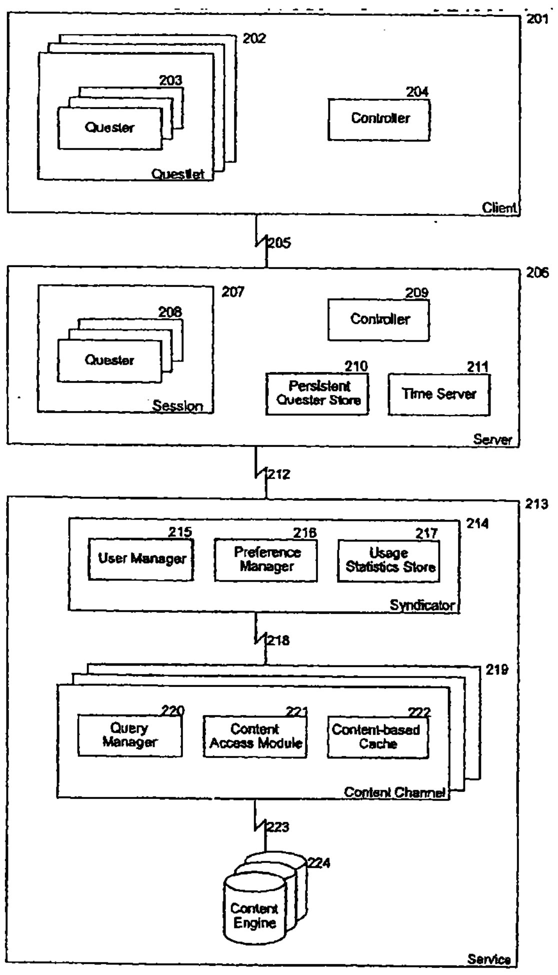
(22) Filed: Aug. 20, 2001

Publication Classification

(51) Int. Cl.^{7} G06F 15/16
(52) U.S. Cl. 709/227; 709/203

(57) ABSTRACT

The invention provides a session-based bi-directional multi-tier client-server asynchronous information database search and retrieval system for sending a character-by-character string of data to an intelligent server that can be configured to immediately analyze the lengthening string character-by-character and return to the client increasingly appropriate database information as the client sends the string.



MO000004

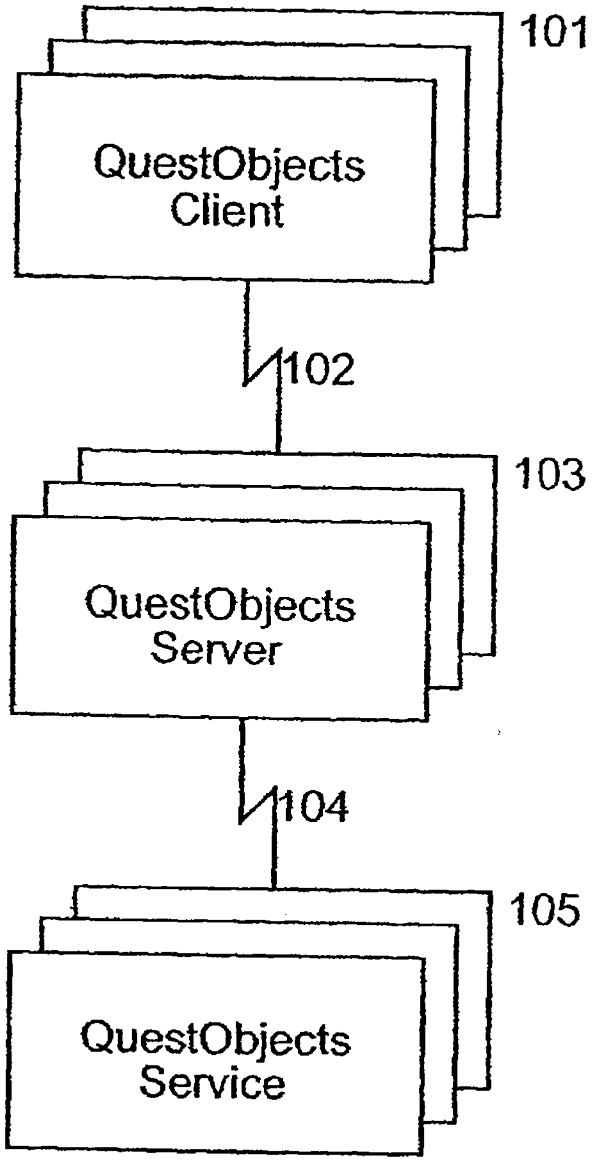

101
QuestObjects Client
102
103
QuestObjects Server
104
105
QuestObjects Service

FIGURE 1

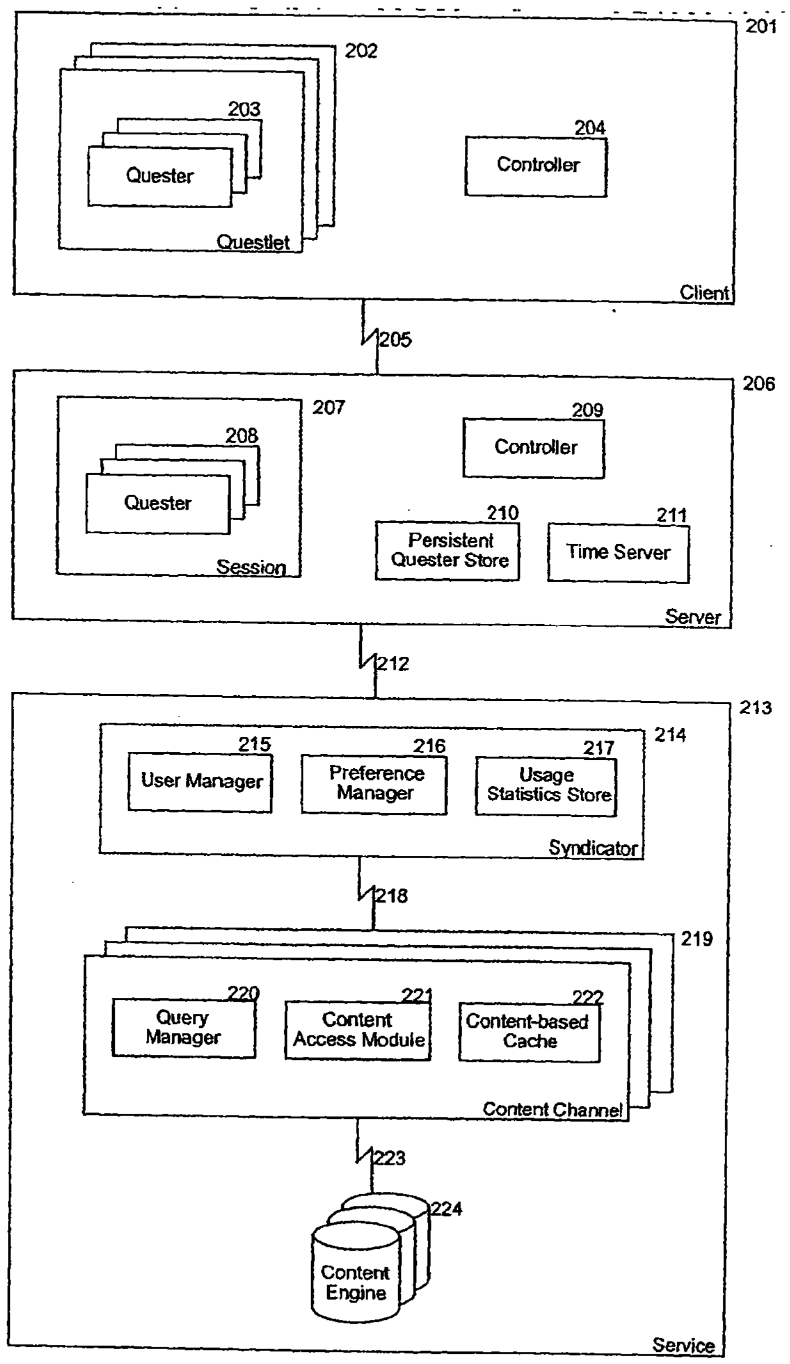

201
202
203
Quester
Questlet
204
Controller
Client
205
206
207
208
Quester
Session
209
Controller
210
Persistent Quester Store
211
Time Server
Server
212
213
214
215
User Manager
216
Preference Manager
217
Usage Statistics Store
Syndicator
218
219
220
Query Manager
221
Content Access Module
222
Content-based Cache
Content Channel
223
224
Content Engine
Service

FIGURE 2

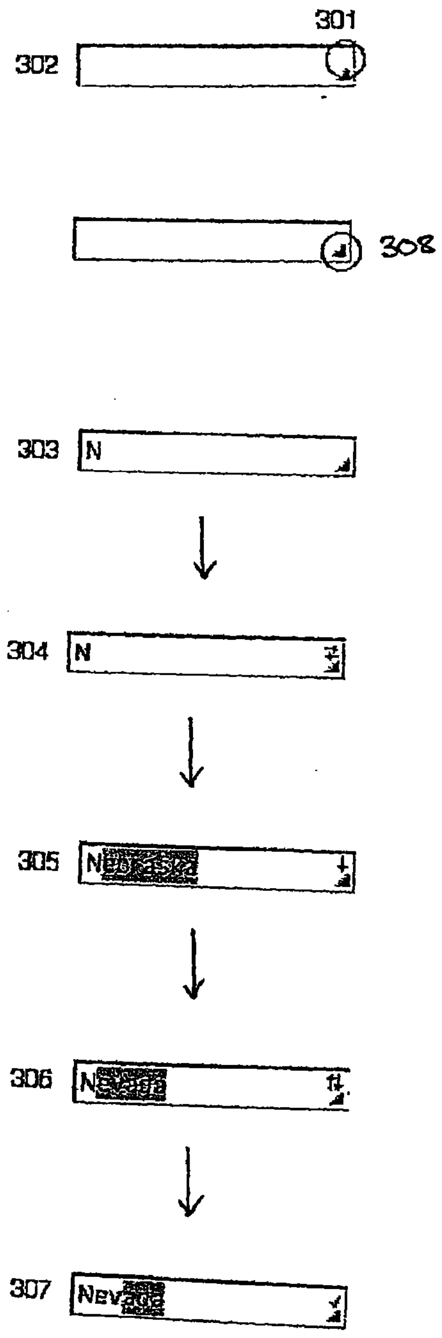


FIGURE 3A

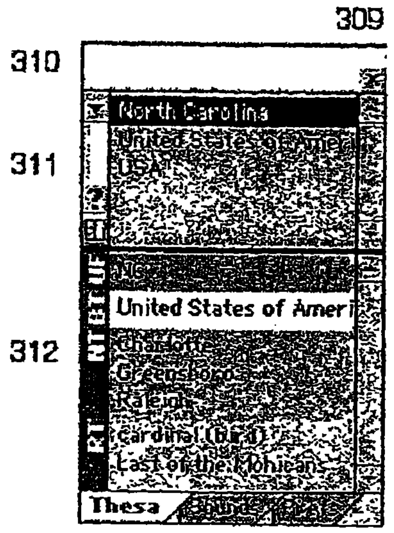


FIGURE 3B

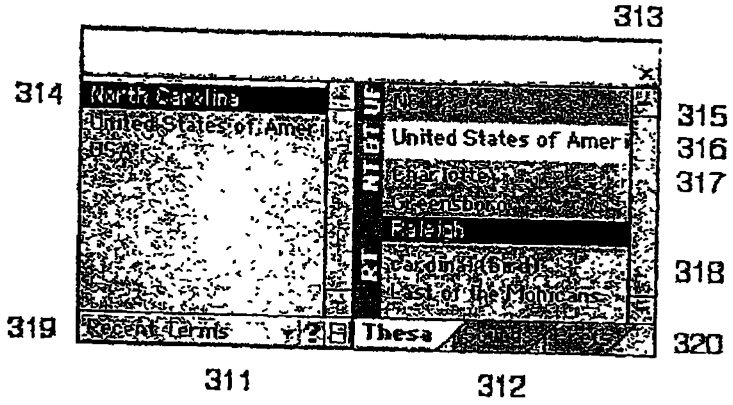


FIGURE 3C

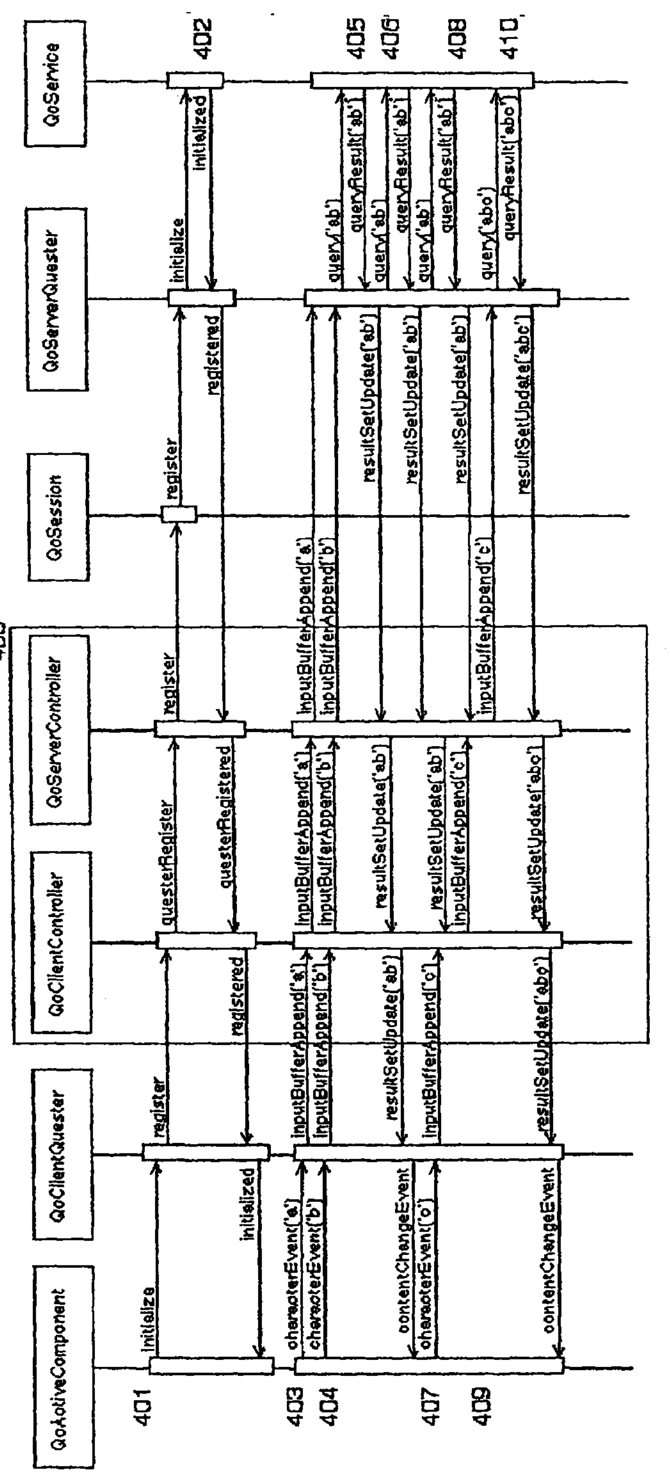

QoActiveComponent
QoClientQuester
QoClientController
QoServerController
QoSession
QoServerQuester
QoService
400
401
402
403
404
405
406
407
408
409
410
initialize
initialized
register
registered
questerRegister
questerRegistered
characterEvent('a')
characterEvent('b')
inputBufferAppend('a')
inputBufferAppend('b')
query('ab')
queryResult('ab')
resultSetUpdate('ab')
contentChangeEvent
characterEvent('c')
inputBufferAppend('c')
query('abc')
queryResult('abc')
resultSetUpdate('abc')

FIGURE 4

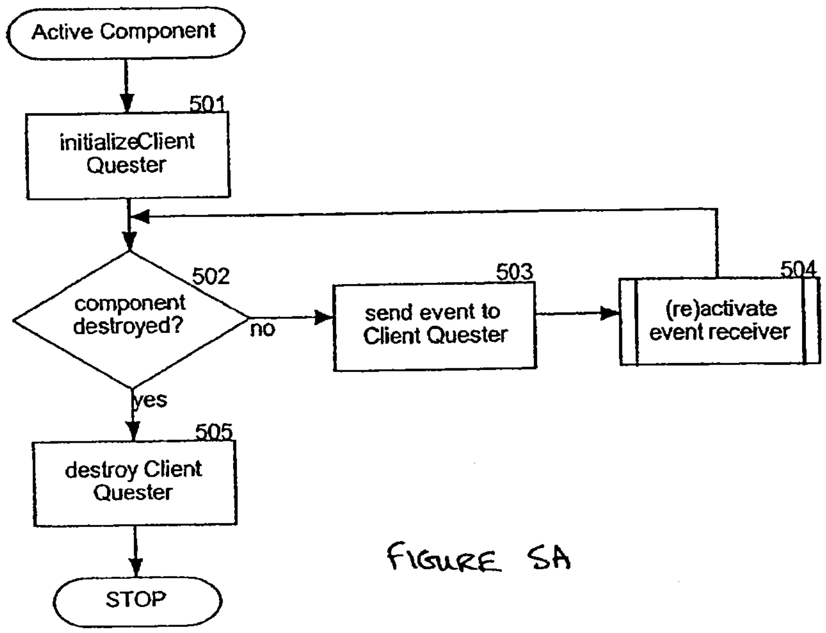


FIGURE 5A

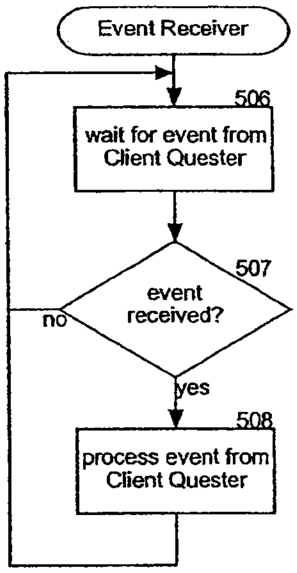

Event Receiver
506
wait for event from Client Quester
507
event received?
no
yes
508
process event from Client Quester

FIGURE 5B

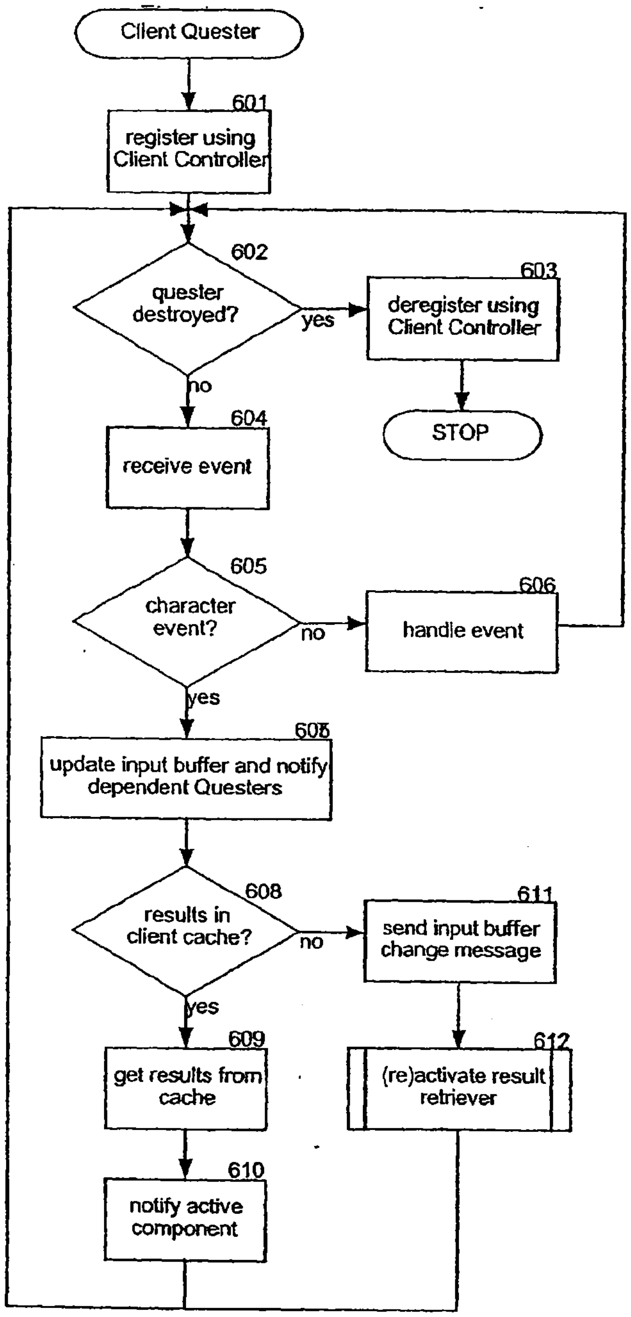

Client Quester
601
register using Client Controller
602
quester destroyed?
yes
603
deregister using Client Controller
STOP
no
604
receive event
605
character event?
no
606
handle event
yes
607
update input buffer and notify dependent Questers
608
results in client cache?
no
611
send input buffer change message
612
(re)activate result retriever
yes
609
get results from cache
610
notify active component

FIGURE 6A

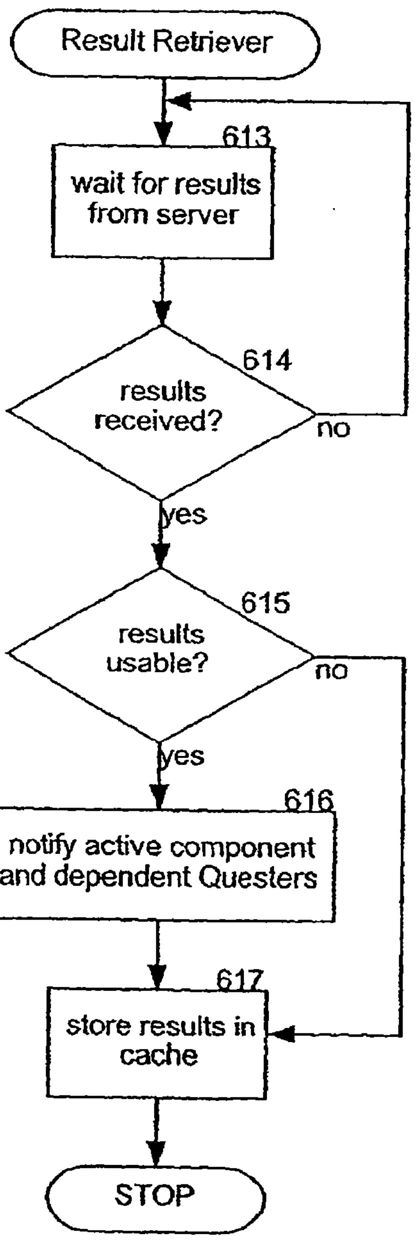

Result Retriever
613
wait for results from server
614
results received?
no
yes
615
results usable?
no
yes
616
notify active component and dependent Questers
617
store results in cache
STOP

FIGURE 6B

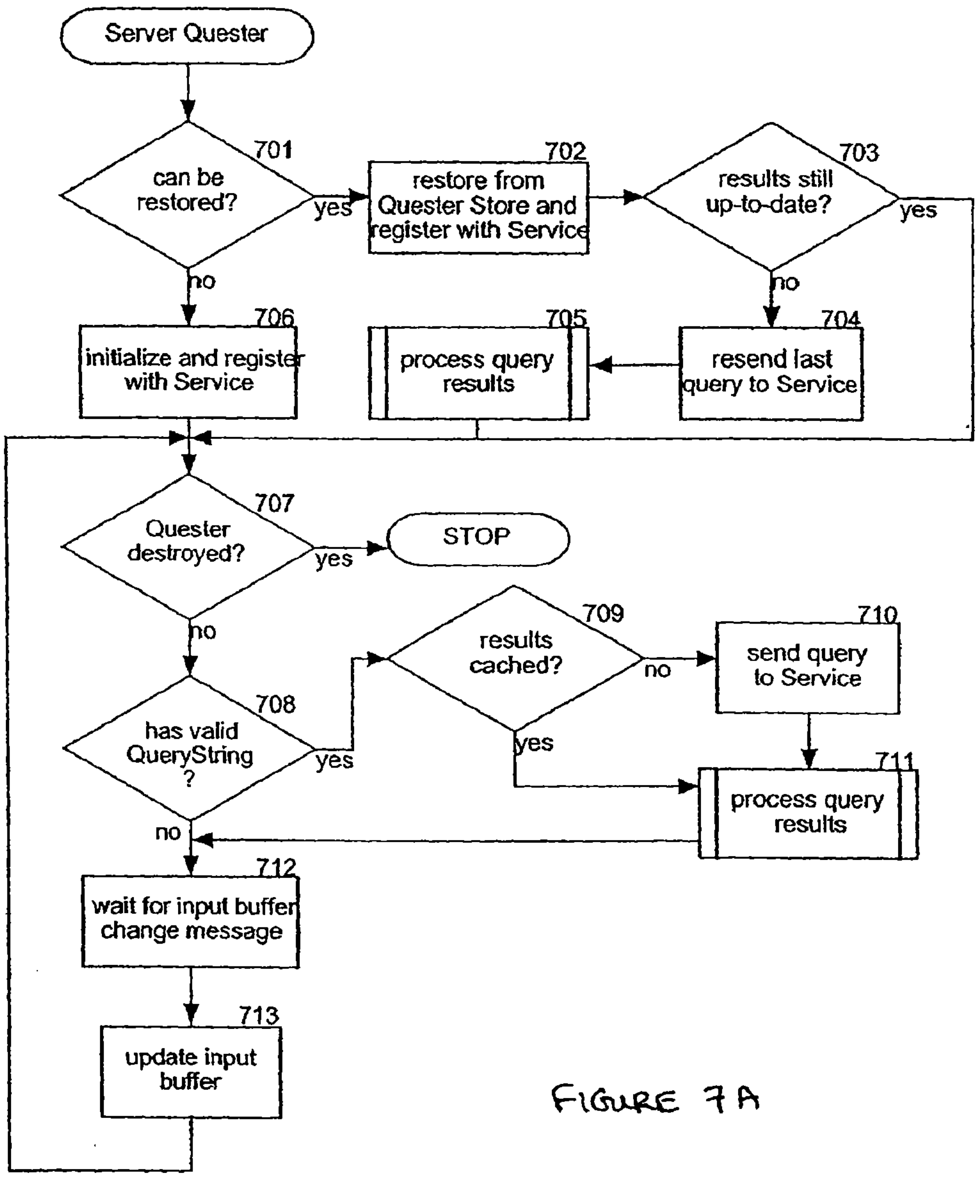

Server Quester
701
can be restored?
yes
no
702
restore from Quester Store and register with Service
703
results still up-to-date?
yes
no
704
resend last query to Service
705
process query results
706
initialize and register with Service
707
Quester destroyed?
yes
STOP
no
708
has valid QueryString ?
yes
no
709
results cached?
no
yes
710
send query to Service
711
process query results
712
wait for input buffer change message
713
update input buffer

FIGURE 7A

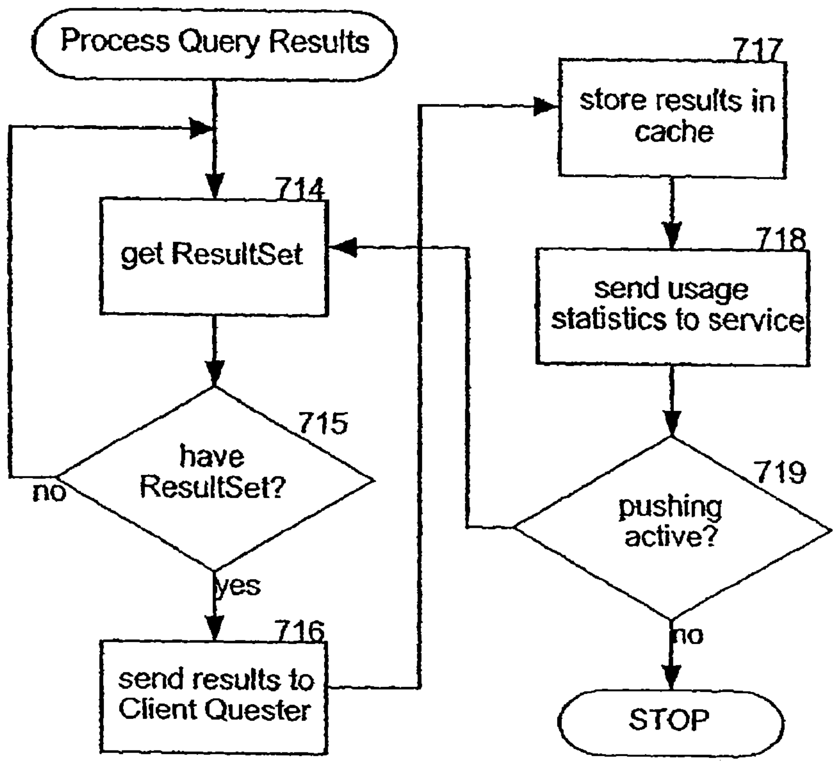

Process Query Results
714
get ResultSet
715
have ResultSet?
no
yes
716
send results to Client Quester
717
store results in cache
718
send usage statistics to service
719
pushing active?
no
STOP

FIGURE 7B

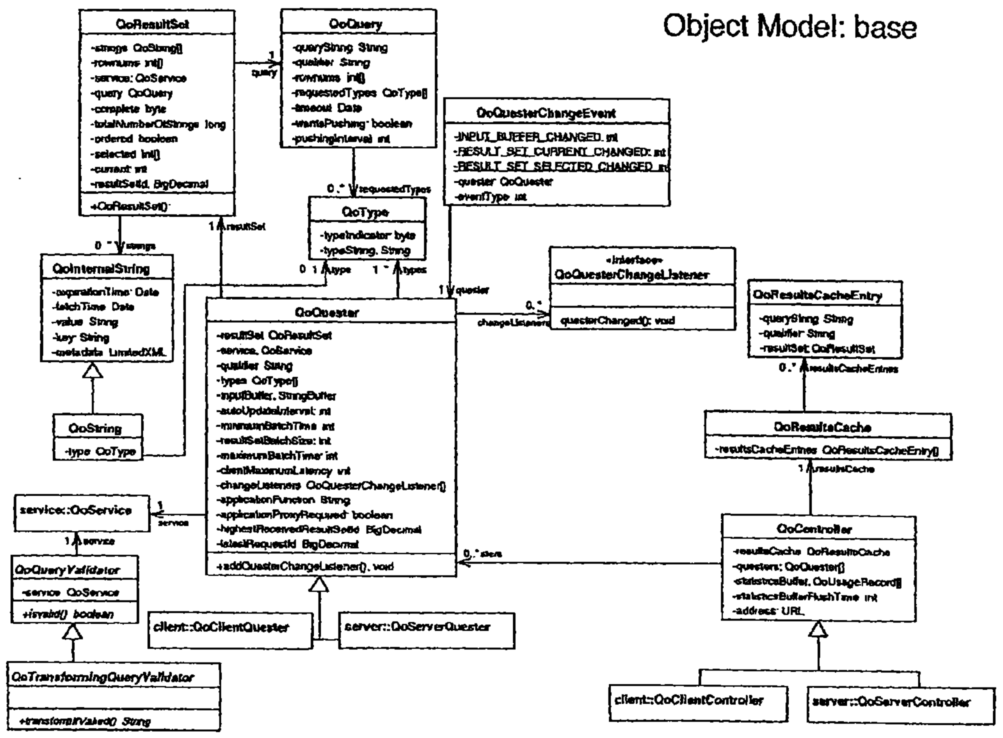

Object Model: base
QoResultSet
-strings: QoString[]
-rownums: int[]
-service: QoService
-query: QoQuery
-complete: byte
-totalNumberOfStrings: long
-ordered: boolean
-selected: int[]
-current: int
-resultSetId: BigDecimal
+QoResultSet()
QoQuery
-queryString: String
-qualifier: String
-rownums: int[]
-requestedTypes: QoType[]
-timeout: Date
-wantsPushing: boolean
-pushingInterval: int
QoQuesterChangeEvent
-INPUT_BUFFER_CHANGED: int
-RESULT_SET_CURRENT_CHANGED: int
-RESULT_SET_SELECTED_CHANGED: int
-quester: QoQuester
-eventType: int
QoType
-typeIndicator: byte
-typeString: String
QoInternalString
-expirationTime: Date
-fetchTime: Date
-value: String
-key: String
-metadata: LimitedXML
QoString
-type: QoType
«interface»
QoQuesterChangeListener
questerChanged(): void
QoResultsCacheEntry
-queryString: String
-qualifier: String
-resultSet: QoResultSet
QoQuester
-resultSet: QoResultSet
-service: QoService
-qualifier: String
-types: QoType[]
-inputBuffer: StringBuffer
-autoUpdateInterval: int
-minimumBatchTime: int
-resultSetBatchSize: int
-maximumBatchTime: int
-clientMaximumLatency: int
-changeListeners: QoQuesterChangeListener[]
-applicationFunction: String
-applicationProxyRequired: boolean
-highestReceivedResultSetId: BigDecimal
-latestRequestId: BigDecimal
+addQuesterChangeListener(): void
QoResultsCache
-resultsCacheEntries: QoResultsCacheEntry[]
service::QoService
QoQueryValidator
-service: QoService
+isValid(): boolean
QoTransformingQueryValidator
+transformIfValid(): String
client::QoClientQuester
server::QoServerQuester
QoController
-resultsCache: QoResultsCache
-questers: QoQuester[]
-statisticsBuffer: QoUsageRecord[]
-statisticsBufferFlushTime: int
-address: URL
client::QoClientController
server::QoServerController
1 query
0..* requestedTypes
0..* strings
1 resultSet
0..1 type
1..* types
1 quester
0..* changeListeners
0..* resultsCacheEntries
1 resultsCache
1 service
1 service
0..* items

FIGURE 8A

Object Model: client

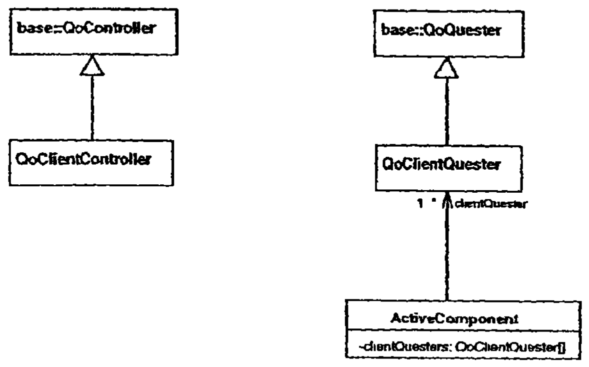


FIGURE 8B

Object Model: server

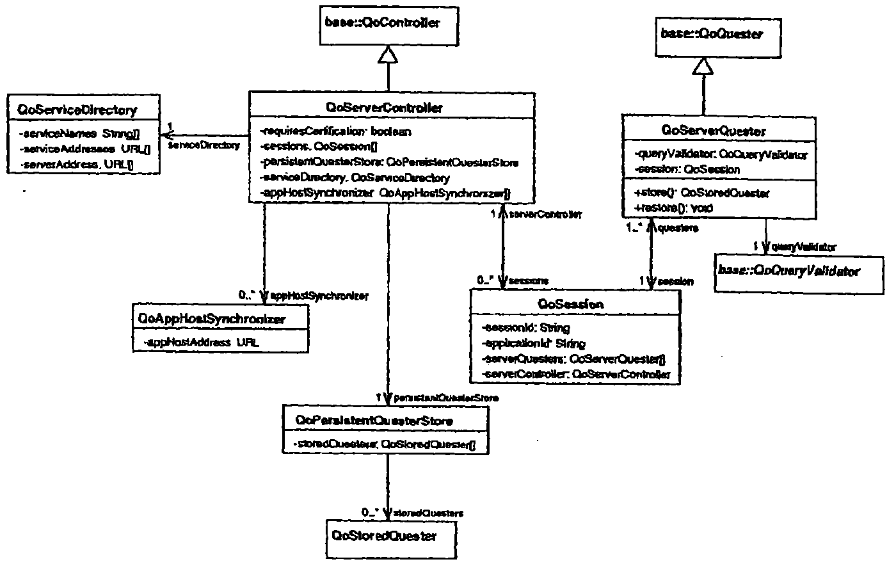


FIGURE 8C

Object Model: service

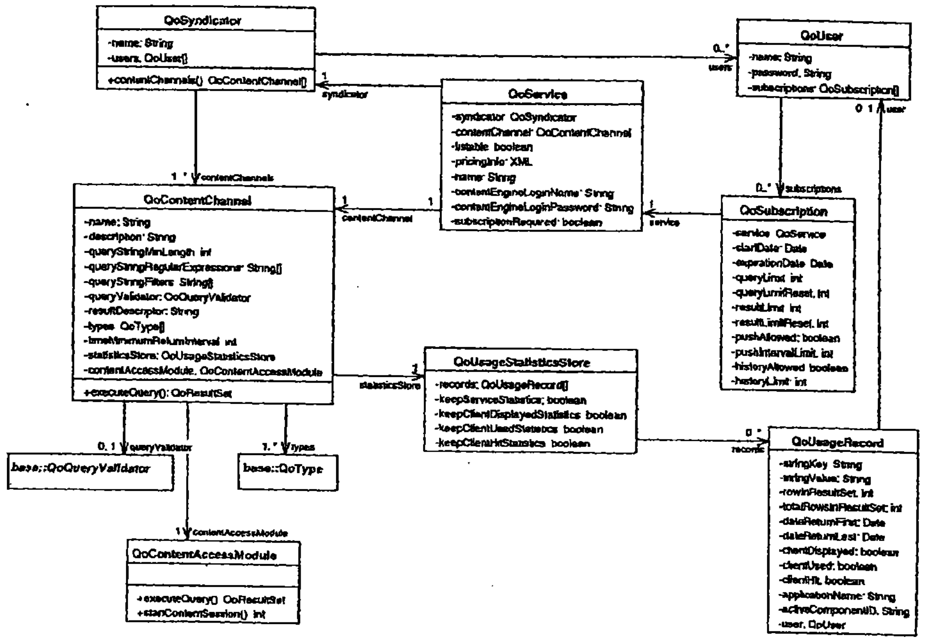


FIGURE 8D

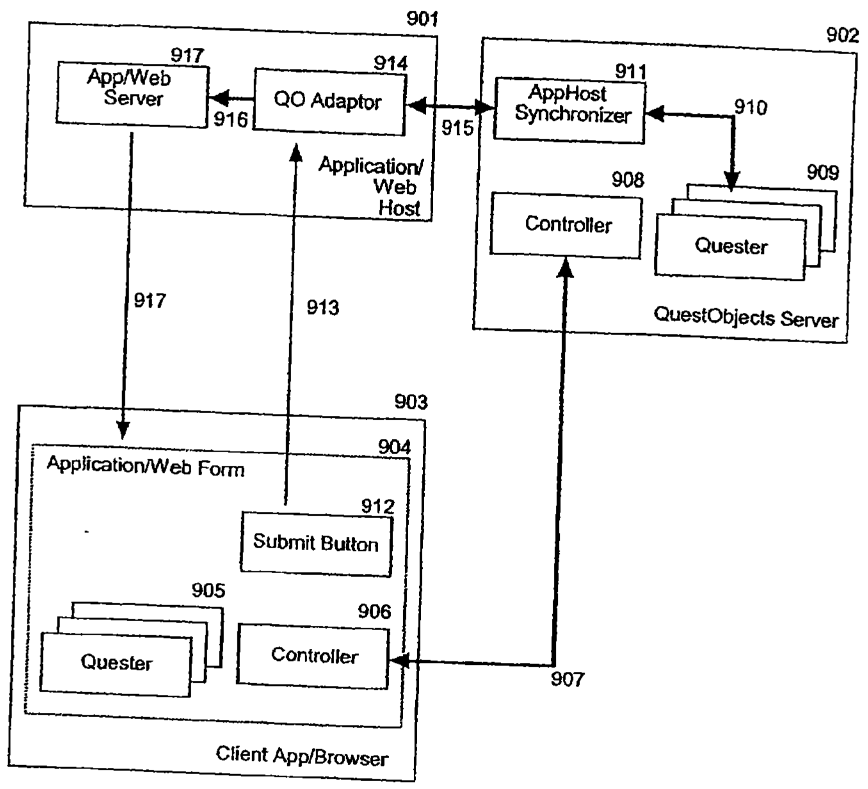

901
917
App/Web Server
914
QO Adaptor
916
915
Application/ Web Host
902
911
AppHost Synchronizer
910
908
Controller
909
Quester
QuestObjects Server
917
913
903
904
Application/Web Form
912
Submit Button
905
Quester
906
Controller
907
Client App/Browser

FIGURE 9

SYSTEM AND METHOD FOR ASYNCHRONOUS CLIENT SERVER SESSION COMMUNICATION

COPYRIGHT NOTICE

FIELD OF THE INVENTION

[0002] The invention relates generally to client-server communication systems, and particularly to a session-based bi-directional multi-tier client-server asynchronous search and retrieval system.

BACKGROUND OF THE INVENTION

[0003] A primary task of computer systems is to manage large quantities of information, generally referred to as data. The first computers typically stored data using off-line methods, for example by using punch cards and other primitive means. As built-in or on-line storage solutions became more affordable, data were instead stored in central memory banks. The first enterprise-wide computer systems consisted of central computers containing central data storage, and a large number of user terminals that accessed this server data by sending input and receiving output as characters to be displayed or printed at the terminal. Although these systems had a primitive user interface and data access became increasingly slower as the number of users grew, these systems nevertheless handled enterprise data with ease and great security.

[0004] The first servers, often referred to as mainframes or mini computers, ran on proprietary operating systems. Terminals usually had large input buffers where input was only checked against or committed to the server after entering text into a page or form. Many systems only displayed the character entered after it was received and confirmed by the server. Faster servers and more modern server operating systems, such as Unix and VMS, offered several advantages in that users could receive immediate feedback after each character was typed.

[0005] At the beginning of the 1980s decade, the growing popularity of microcomputers and personal workstations made it possible to store data locally. Enterprise data was distributed over networks of computer systems. To access information it was no longer necessary to have a continuous connection to central databases, and instead it was possible to copy information to a personal computer, edit and work with it, and then save it back to a file or database server later. Most microcomputers worked with data in logical chunks or files. This brought a lot of power to end users, but introduced problems in managing the large quantity of enterprise data that was no longer stored as a unique entity in one place. For example, a file that was being edited by one user could not usually be accessed or modified by other users at the same time. It was also difficult to manage multiple copies of the same data.

[0006] Toward the end of the 1980's faster microcomputers and networks made it practical to work with enterprise data in smaller chunks than files. One example of this new technology was the development of Structured Query Language (SQL) relational databases which made it possible to divide software programs into a 'Client' tier and a 'Server' tier, that communicated with each other over a network. Client-server computing thus made it possible to store information centrally, yet manage and work with it locally. In the client-server paradigm, the client systems concentrated on offering a user-friendly interface to server data, while the server systems were able to handle many client systems at once while safely managing enterprise data.

[0007] However, the increasing client-server computing introduced its share of problems. Protocols used to communicate between client and server became increasingly complex and difficult to manage. Enterprise IT departments needed increasingly greater resources to manage the proprietary implementations of client operating systems, server database systems and middleware protocols connecting the various 'tiers' of client-server systems. Data was no longer stored in one place but was required to be managed within a distributed network of systems. Client-server systems also lacked a major advantage of mainframes: in a client-server system any changes to the data on the server weren't immediately updated on the client.

[0008] Starting in the 1990s, the Internet has allowed businesses, organizations, and other enterprises to easily make information available to users without the complex architecture that client-server systems typically require. Today, an increasing number of software applications are moving their data and logic or functional processes back to the server tier, from which they can be accessed from the Internet by a wide variety of clients, including thin and very thin-clients, which typically consist of Internet browsers or small applications (applets) whose sole responsibility is providing an interface to the user. In many ways, Internet computing (often referred to as e-commerce) has brought back the data-handling advantages of mainframes. Within the e-commerce environment data that change on the server are immediately available to clients that access the data through the Internet (world-wide) or through an intranet (enterprise-wide).

[0009] Unfortunately, the rise of Internet commerce has also given rise to some of the disadvantages associated with mainframe technology. Most Internet connections that present data to the user or client process use the Hyper Text Transfer Protocol (HTTP) which is inherently "session-less." This means that, for example, there is no totally reliable way for the server to automatically update the client display once the server data change. It also means that the server only checks the validity of the client or user input after the user sends back or submits an entire input form. This apparent disadvantage has also played an important role in the success of the Internet: because HTTP connections are session-less, they require much less processing power and much less memory on the server while the user is busy entering data. Thus, Internet applications running on web servers can be accessed by millions of people. Because HTTP and related Internet-based client-server systems do not provide continuous access to server data, systems sometimes incorporate lookup tables and pre-defined values that are cached locally. For example, a list of possible countries to be selected by a user of a web page can be sent to the user's computer when that page is first sent to the user and used thereafter for subsequent country selections. Client-server applications often pre-read the data from the server the moment an application or application window is opened, in order to present users with selection lists the moment they need them. This poses problems for data that frequently changes overtime since the client system may allow users to select or enter data that is no longer valid. It also poses problems for large selection lists whose transmission to the client may take a long time.

[0010] To address this some systems incorporate a local cache of the data frequently accessed by the user. A web browser may, for example be configured to remember the last pages a user visited by storing them in a local cache file. A clear disadvantage of keeping such a local cache is that it is only useful as long as the user stays on the same client computer system. Also, the local cache may include references to web pages that no longer exist.

[0011] Some other systems with limited network bandwidth (like cell phones or personal organizers) can be deployed with built-in databases (such as dictionaries and thesauri), because it would be impractical to wait for the download of an entire database, which is needed before the data is of any use. This has the disadvantage that data stored in the device may no longer be up-to-date because it's really a static database. Also, the cost of cell phones and personal organizers is greatly increased by the need for megabytes of local storage. Another important consideration is that keeping valuable data in any local database makes it vulnerable to misuse and theft. What is needed is a mechanism that addresses these issues that allows a client-server system to retain some element of a session-based system, with its increase in performance, while at the same time offering a secure communication mechanism that requires little, if any, local storage of data.

[0012] Other attempts have been made to tackle some of the problems inherent with traditional computer system interfaces, and particularly with regard to user session administration and support. These attempts include the auto-complete function systems such as used in Microsoft Internet Explorer, the spell-as-you-go systems such as found in Microsoft Word, and the wide variety of client-server session managers such as Netopia's Timbuktu and Citrix Winframe.

[0013] Auto-Complete Functionality

[0014] Many current systems provide a mechanism to auto-complete words entered into fields and documents. This 'auto-complete' functionality is sometimes called 'type-ahead' or 'predictive text entry'. Many web browsers such as Microsoft's Internet Explorer application will automatically 'finish' the entry of a URL, based on the history of web sites visited. E-mail programs including Microsoft Outlook will automatically complete names and e-mail addresses from the address book and a history of e-mails received and sent. Auto-completion in a different form is found in most graphical user interfaces, including operating systems such as Microsoft Windows and Apple Mac OS, that present lists to the user: When the user types the first character of a list entry, the user interface list will automatically scroll down to that entry. Many software development tools will automatically complete strings entered into program source code based on a known taxonomy of programming-language dependent keywords and 'function names' or 'class names' previously entered by the developer. Some cell phones and personal organizers also automatically type-ahead address book entries or words from a built-in dictionary. Auto-complete functionality facilitates easy entry of data based on prediction of what options exist for the user at a single moment in time during entry of data.

[0015] Checking As You Go

[0016] More and more word processing programs (most notably Microsoft Word and certain e-mail programs) include so-called 'spell checking as you type'. These programs automatically check the spelling of words entered while the user is typing. In a way, this can be seen as 'deferred auto-complete', where the word processor highlights words after they were entered, if they don't exist in a known dictionary. These spell checking programs often allow the user to add their own words to the dictionary. This is similar to the 'history lists' that are maintained for the auto-completion of URLs in a web browser, except that in this case the words are manually added to the list of possible 'completions' by the user.

[0017] Software Component Technologies

[0018] Software component technologies have provided a measure of component generation useful in client/server systems. One of these technologies is OpenDoc, a collaboration between Apple Computer, Inc. and IBM Corporation (amongst others) to allow development of software components that would closely interact, and together form applications. One of the promises of OpenDoc was that it would allow small developers to build components that users could purchase and link together to create applications that do exactly what the users want, and would make existing 'bloat-ware' applications (notably Microsoft Office and Corel's WordPerfect Office/Corel Office) redundant, but the technology was dropped several years ago in favor of newer technologies such as CORBA (Common Object Request Broker Architecture), developed by the Object Management Group to allow transparent communication and interoperability between software components.

[0019] Object-oriented languages and even non-object-oriented (database) systems have used component technologies to implement technical functionality. The NeXTstep operating system from NeXT Computer, Inc. (which was later acquired by Apple Computer, Inc. and evolved into the Mac operating system Mac OS X) had an object-oriented architecture from its original beginnings, that allowed software developers to create applications based on predefined, well-tested and reliable components. Components could be 'passive' user interface elements (such as entry fields, scroll areas, tab panes etc) used in application windows. But components could also be active and show dynamic data (such as a component displaying a clock, world map with highlight of daylight and night, ticker tape showing stock symbols, graphs showing computer system activity, etc.). The NeXT operating system used object frameworks in the Objective C language to achieve its high level of abstraction which is needed for components to work well. Later, Sun Microsystems, Inc. developed the Java language specification in part to achieve the same goal of interoperability. To date, Java has probably been the most successful 'open' (operating system independent) language used to build software components. It is even used on certain web sites that allow 'Java applets' on the user's Internet browser to continuously show up-to-date information on the client system.

[0020] WebObjects, an object-oriented technology developed by Apple Computer, Inc. is an Internet application server with related development tools, which was first developed by NeXT Computer, Inc. WebObjects uses object oriented frameworks that allow distribution of application logic between server and client. Clients can be HTML-based, but can also be Java applets. WebObjects uses proprietary technology that automatically synchronizes application objects between client and server. The layer that synchronizes data objects between the client and the server is called the 'Enterprise Object Distribution' (EODistribution), part of Apple's Enterprise Objects Framework (EOF), and is transparent to the client software components and the server software components.

[0021] Session Management

[0022] Both Netopia's Timbuktu remote access systems, and Citrix, Inc.'s Winframe terminal server product, allow some element of remote access to server applications from a client system. These products synchronize user data and server data, transparently distributing all user input to the server and return all server(display) output to the client. Timbuktu does this with very little specific knowledge about the application and operating system used. This allows it to transparently work on both Microsoft Windows and Mac OS platforms. Technologies similar to Timbuktu do exist and perform the same kind of 'screen sharing'. For example, the Virtual Network Computing (VNC) system is one example of an open source software program that achieves the same goals and also works with Linux and Unix platforms.

[0023] Citrix Winframe has taken the same idea a step further by incorporating intimate knowledge of the Microsoft Windows operating system (and its Win32 APIs) to further optimize synchronization of user input and application output on the server. It can then use this detailed knowledge of the Microsoft Windows APIs to only redraw areas of the screen that it knows will change based on a user action: for example, Winframe may redraw a menu that is pulled down by the user without needing to access the server application because it knows how a menu will work.

[0024] Software Applications

[0025] Several application providers have also built upon these technologies to provide applications and application services of use to the end-user. These applications include computer-based thesaurii, on-line media systems and electronic encyclopediae.

[0026] The International Standards Organization (as detailed further in ISO 2788—1986 Documentation—Guidelines for the Establishment and Development of monolingual thesauri and ISO 5964—1985 Documentation—Guidelines for the Establishment and Development of multilingual thesauri) determines suggested specifications for electronic thesauri, and thesaurus management software is now available from numerous software vendors worldwide. However, most systems have clear limitations that compromize their user-friendliness. Most commonly this is because they use a large third-party database system, such as those from Oracle Software, Inc. or Informix, Inc. as a back-end database. This means that any thesaurus terms that are displayed to the user are fetched from the database and then presented in a user interface. If one user changes the contents of the thesaurus, other users will only notice that change after re-fetching the data. While of little concern in small or infrequently changing environments, this problem is a considerable one within larger organizations and with rapidly updated content changes, for example in media publishing applications when thesaurus terms are being linked to new newspaper or magazine articles. This type of work is usually done by multiple documentalists (media content authors) simultaneously. To avoid 'mixing up' terms linked to articles, each documentalist must be assigned a certain range of articles to 'enrich' (which in one instance may be the act of adding metadata and thesaurus terms to a document). Clearly, in these situations there is a great need for live updates of data entered by these users, but a similar need exists for all client-server database programs.

SUMMARY OF THE INVENTION

[0027] The invention provides a system that offers a highly effective solution to the aforementioned disadvantages of both client-server and Internet systems by providing a way to synchronize the data entered or displayed on a client system with the data on a server system. Data input by the client are immediately transmitted to the server, at which time the server can immediately update the client display. To ensure scalability, systems built around the present invention can be divided into multiple tiers, each tier being capable of caching data input and output. A plurality of servers can be used as a middle-tier to serve a large number of static or dynamic data sources, herein referred to as "content engines."

[0028] The present invention may be incorporated in a variety of embodiments to suit a correspondingly wide variety of applications. It offers a standardized way to access server data that allows immediate user-friendly data feedback based on user input. Data can also be presented to a client without user input, i.e. the data are automatically pushed to the client. This enables a client component to display the data immediately, or to transmit the data to another software program to be handled as required.

[0029] The present invention can also be used to simply and quickly retrieve up-to-date information from any string-based content source. Strings can be linked to metadata allowing user interface components to display corresponding information such as, for example, the meaning of dictionary words, the description of encyclopedia entries or pictures corresponding to a list of names.

[0030] Embodiments of the present invention can be used to create a user interface component that provides a sophisticated "auto-completion" or "type-ahead" function that is extremely useful when filling out forms. This is analogous to simple, client-side auto-complete functions that have been widely used throughout the computing world for many years. As a user inputs data into a field on a form, the auto-complete function analyzes the developing character string and makes intelligent suggestions about the intended data being provided. These suggestions change dynamically as the user types additional characters in the string. At any time, the user may stop typing characters and select the appropriate suggestion to auto-complete the field.

[0031] Today's client-side auto-complete functions are useful but very limited. The invention, however, vastly expands the usefulness and capabilities of the auto-complete function by enabling the auto-complete data, logic and intelligence to reside on the server, thus taking advantage of server-side power. Unlike the client-side auto-complete functions in current use, an auto-complete function created by the present invention generates suggestions at the server as the user types in a character string. The suggestions may be buffered on a middle tier so that access to the content engine is minimized and speed is optimized.

[0032] The simple auto-complete schemes currently in popular use (such as email programs that auto-complete e-mail addresses, web browsers that auto-complete URLs, and cell phones that auto-complete names and telephone numbers) require that the data used to generate the suggestions be stored on the client. This substantially limits the flexibility, power, and speed of these schemes. The present invention, however, stores and retrieves the auto-complete suggestions from databases on the server. Using the present invention, the suggestions generated by the server may, at the option of the application developer, be cached on the middle tier or on the client itself to maximize performance.

[0033] The present invention provides better protection of valuable data than traditional methods, because the data is not present on the client until the moment it is needed, and can be further protected with the use of user authentication, if necessary.

[0034] The present invention is also useful in those situations that require immediate data access, since no history of use needs to be built on the client before data is available. Indeed, data entered into an application by a user can automatically be made available to that user for auto-completion on any other computer, anywhere in the world.

[0035] Unlike existing data-retrieval applications, server data can be accessed through a single standardized protocol that can be built into programming languages, user interface components or web components. The present invention can be integrated into and combined with existing applications that access server data. Using content access modules, the present invention can access any type of content on any server.

[0036] In the detailed description below, the present invention is described with reference to a particular embodiment named QuestObjects. QuestObjects provides a system for managing client input, server queries, server responses and client output. One specific type of data that can be made available through the system from a single source (or syndicate of sources) is a QuestObjects Service. Other terms used to describe the QuestObjects system in detail can be found in the glossary given below.

[0037] QuestObjects is useful for retrieval of almost any kind of string-based data, including the following QuestObjects Service examples:

INTRANET USE

[0038] Access system for database fields (for lookup and auto-complete services).

[0039] Enterprise thesauri system.

[0040] Enterprise search and retrieval systems.

[0041] Enterprise reference works.

[0042] Enterprise address books.

[0043] Control systems for sending sensor readings to a server that responds with appropriate instructions or actions to be taken.

INTERNET USE

[0044] Client access to dictionary, thesaurus, encyclopedia and reference works.

[0045] Access to commercial products database.

[0046] Literary quotes library.

[0047] Real-time stock quote provision.

[0048] Access to real-time news service.

[0049] Access to Internet advertisements.

[0050] Access to complex functions (bank check, credit card validation, etc).

[0051] Access to language translation engines.

[0052] Access to classification schemes (eg, Library of Congress Subject Headings).

[0053] Access to lookup lists such as cities or countries in an order form.

[0054] Personal address books.

[0055] Personal auto-complete histories.

BRIEF DESCRIPTION OF THE FIGURES

[0056] **FIG. 1** shows a general outline of a system incorporating the present invention.

[0057] **FIG. 2** shows a schematic of a system in accordance with an embodiment of the invention.

[0058] **FIG. 3A** shows a variety of stages in the usage of a sample Questlet implementation in accordance with an embodiment of the invention.

[0059] **FIG. 3B** shows an expanded view of a sample Questlet implementation in accordance with an embodiment of the invention.

[0060] **FIG. 3C** shows an expanded view of a sample Questlet implementation in accordance with an embodiment of the invention.

[0061] **FIG. 4** shows a sequence diagram illustrating the use of a system in accordance with an embodiment of the invention.

[0062] **FIG. 5A** shows a first thread flow chart illustrating the interface between an active component and an embodiment of the invention.

[0063] **FIG. 5B** shows a second thread flow chart illustrating the interface between an active component and an embodiment of the invention.

[0064] **FIG. 6A** shows a first thread flow chart illustrating the client side of an embodiment of the invention.

[0065] **FIG. 6B** shows a second thread flow chart illustrating the client side of an embodiment of the invention.

[0066] **FIG. 7A** shows a first thread flow chart illustrating the server side of an embodiment of the invention.

[0067] FIG. 7B shows a second thread flow chart illustrating the server side of an embodiment of the invention.

[0068] FIG. 8A shows an object model of an embodiment of the present invention, displaying the base part.

[0069] FIG. 8B shows an object model of an embodiment of the present invention, displaying the client part.

[0070] FIG. 8C shows an object model of an embodiment of the present invention, displaying the server part.

[0071] FIG. 8D shows an object model of an embodiment of the present invention, displaying the service part.

[0072] FIG. 9 shows a schematic of an application proxy system that enables the use of the invention in various client environments.

DETAILED DESCRIPTION

[0073] Roughly described, the invention provides a session-based bi-directional multi-tier client-server asynchronous information database search and retrieval system for sending a character-by-character string of data to an intelligent server that can be configured to immediately analyze the lengthening string character-by-character and return to the client increasingly appropriate database information as the client sends the string.

[0074] The present invention includes a system that offers a highly effective solution to an important disadvantage of both client-server and Internet systems: The present invention provides a standardized way to immediately synchronize the data entered or displayed on a client system with the data on a server system. Data input by the client is immediately transmitted to the server at which time the server can immediately update the client display. To ensure scalability, systems built around the present invention can be divided into multiple 'tiers' each capable of caching data input and output. Any number of servers can be used as a middle-tier to serve any number of static or dynamic data sources (often referred to as "Content Engines").

[0075] The present invention is useful for an extremely wide variety of applications. It offers a standardized way to access server data that allows immediate user-friendly data feedback based on user input. Data can also be presented to a client without user input, i.e. the data is automatically 'pushed' to the client. This enables a client component to display the data immediately or to transmit it to another software program to be handled as required.

[0076] The present invention is also particularly useful for assistance in data entry applications, but can also be used to simply and quickly retrieve up-to-date information from essentially any string-based content source. Strings can be linked to metadata allowing user interface components to display corresponding information such as the meaning of dictionary words, the description of encyclopedia entries or pictures corresponding to a list of names.

[0077] In some embodiments, the present invention can be used to create a user interface component that provides a sophisticated "auto-completion" or "type-ahead" function that is extremely useful when filling out forms. Simple, client-side auto-complete functions have been widely used throughout the computing world for many years. As a user inputs data into a field on a form, the auto-complete function analyzes the developing character string and makes "intelligent" suggestions about the intended data being provided. These suggestions change dynamically as the user types additional characters in the string. At any time, the user may stop typing characters and select the appropriate suggestion to auto-complete the field.

[0078] Today's client-side auto-complete functions are very limited. The present invention vastly expands the usefulness and capabilities of the auto-complete function by enabling the auto-complete data, logic and intelligence to reside on the server thus taking advantage of server-side power. Unlike the client-side auto-complete functions in current use, an auto-complete function created by the present invention pushes suggestions from the server as the user types in a character string. Using the present invention, the suggestions may be buffered on a middle tier so that access to the content engine is minimized and speed is optimized.

[0079] The simple auto-complete schemes currently in popular use (such as email programs that auto-complete e-mail addresses, web browsers that auto-complete URLs, and cell phones that auto-complete names and telephone numbers) require that the data used to generate the suggestions be stored on the client. This substantially limits the flexibility, power, and speed of these schemes. The present invention, however, stores and retrieves the auto-complete suggestions from databases on the server. Using the present invention, the suggestions generated by the server may, at the option of the application developer, be cached on the middle tier or one the client itself to maximize performance.

[0080] The present invention provides better protection of valuable data because the data is not present on the client until the moment it is needed and can be further protected with a user authentication mechanism, if necessary.

[0081] The present invention is useful for immediate data use, since no use history must be built on the client before data is available. Indeed, data entered into an application by a user can automatically be made available to that user for auto-completion on any other computer anywhere in the world.

[0082] Unlike existing data-retrieval applications, server data can be accessed through a single standardized protocol that can be built into programming languages, user interface components or web components. The present invention can be integrated into, and combined with, existing applications that access server data. Using Content Access Modules, the present invention can access any type of content on any server.

[0083] In the detailed description below, an embodiment of the present invention is referred to as QuestObjects, and provides a system of managing client input, server queries, server responses and client output. One specific type of data made available through the system from a single source (or syndicate of sources) is referred to as a QuestObjects Service. Other terms used to describe the QuestObjects system in detail can be found in the glossary below:

GLOSSARY

[0084] Active Component—Part of a software program that accesses the QuestObjects system through one or more Questers. Active Components may provide a user interface, in which case they're referred to as Questlets.

[0085] AppHost Synchronizer—Part of the QuestObjects Server that allows the Application Proxy access to data in Server Questers.

[0086] Application Proxy—An optional method implemented by the QuestObjects Server allowing the use of the QuestObjects system in client systems that do not allow the

[0087] QuestObjects—Client components to communicate to the application server or web server directly. Uses the AppHost Synchronizer on the QuestObjects Server to send selected strings and metadata to the application server or web server using a QuestObjects Adaptor.

[0088] Client Controller—A QuestObjects Controller on a QuestObjects Client.

[0089] Client Quester—A Quester on a QuestObjects Client that has a Server Quester as its peer.

[0090] ClientSession—A temporary container of information needed to manage the lifespan of Server Questers in a QuestObjects Server.

[0091] Content Access Module—A part of a Content Channel that provides a standardized API to access specific types of Content Engines.

[0092] Content-based Cache—A persistent store of Queries and corresponding Result Sets executed by a Content Engine for a specific Content Channel.

[0093] Content Channel—A part of the QuestObjects system that provides one type of information from one Content Engine. Consists of a Query Manager and a Content Access Module, linking a Content Engine to the QuestObjects system.

[0094] Content Engine—A dynamic data source that provides data to a Content Channel by accessing its own database or by querying other information systems.

[0095] Query Filter—A filter specified by a Query Manager in a specific Service used to tell the Server Quester to interpret incoming strings before they are sent to the Service as a QuestObjects Query.

[0096] Query Manager—An intelligent part of a Content Channel that interprets QuestObjects Queries and sends them to a Content Engine (through a Content Access Module) or retrieves results from the Content-based Cache in a standardized way. The Query Manager can also send a list of Query Patterns and Query Filters to the Server Quester, allowing the Server Quester to match and filter new Queries before they are sent to the Content Channel.

[0097] Query Pattern—A string-matching pattern (such as a unix-style grep pattern) specified by a Query Manager in a specific Service used to tell the Server Quester to interpret incoming strings before they are sent to the Service as a QuestObjects Query.

[0098] Persistent Quester Store—A dynamic database of Questers that is maintained on the QuestObjects Server, allowing Questers to be stored across Client sessions whereby the state and contents of the Client are automatically restored when a new Client Session is started.

[0099] Quester—An intelligent non-visual object contained by an Active Component that links a QuestObjects StringList to an input buffer. Questers exist on both the QuestObjects Client and the QuestObjects Server and can be specifically referred to as Client Quester and Server Quester. Questers communicate with each other through a QuestObjects Controller.

[0100] Questlet—A User Interface Element that accesses the QuestObjects system through one or more Questers. A visual Active Component.

[0101] QuestObjects Adaptor—An optional software component for existing application servers and web servers that allows these servers to use data entered into the QuestObjects system by users of client systems and web browsers that require an Application Proxy.

[0102] QuestObjects Client—Part of the QuestObjects system that functions as the client tier consisting of one or more Client Questers and a Client Controller that communicates to a QuestObjects Server.

[0103] QuestObjects Controller—An intelligent non-visual component that provides the interface between Questers on QuestObjects Clients and QuestObjects Servers. QuestObjects Controllers implement the protocol of the present invention.

[0104] QuestObjects Query—A string created by the Server Quester with optional qualifier and the requested row numbers forming a query to be executed by a specified QuestObjects Service.

[0105] QuestObjects Result Set—A set of StringLists with corresponding Query returned from the QuestObjects Service, returned in batches to the Client Quester by the Server Quester.

[0106] QuestObjects Server—Central part of the QuestObjects system that provides the link between any number of QuestObjects Clients, any number of QuestObjects Services, and any number of other QuestObjects Servers. Maintains Client Sessions that QuestObjects Clients communicate with through the Server Controller. Provides services such as caching, replication and distribution.

[0107] QuestObjects Service—One of the Content Channels provided by a specific Syndicator. A logical name for a Syndicator, a Content Channel and its corresponding Content Engine.

[0108] QuestObjects String—Sequence of Unicode characters with standardized attributes used by the QuestObjects system.

[0109] QuestObjects StringList—Container for a set of QuestObjects Strings retrieved from a QuestObjects Service with standardized attributes needed by the QuestObjects System.

[0110] QuestObjects User—Person or process accessing the QuestObjects system from the QuestObjects Client, optionally authorized by the Syndicator.

[0111] Server Controller—A QuestObjects Controller on a QuestObjects Server.

[0112] Server Quester—A Quester on a QuestObjects Server that has a Client Quester as its peer.

[0113] Syndicator—A part of the QuestObjects system that offers one or more Content Channels to be used by QuestObjects Servers, performing user-based accounting services based on actual data use such as billing, collection of statistics and management of preferences.

[0114] User Interface Element—A visual and optionally interactive component in a software program that provides an interface to the user.

[0115] The present invention provides a system that allows clients or client applications to asynchronously retrieve database information from a remote server of server application. The terms "client" and "server" are used herein to reflect a specific embodiment of the invention although it will be evident to one skilled in the art that the invention may be equally used with any implementation that requires communication between a first process or application and a second process or application, regardless of whether these processes comprise a typical client-server setup or not. The invention includes a Server, that handles requests for information from clients, and a communication protocol that is optimized for sending single characters from a Client to the Server, and lists of strings from the Server to the Client. In one embodiment, as the Server receives a single character from the Client, it immediately analyzes the lengthening string of characters and, based on that analysis, returns database information to the Client in the form of a list of strings. Clients are not restricted to programs with a user interface. Generally, any process or mechanism that can send characters and receive string lists can be considered a client of the system. For example, in an industrial or power supply setting, the control system of a power plant could send sensor readings to the system, and in return receive lists of actions to be taken, based on those sensor readings.

[0116] The system's protocol is not restricted to sending single characters. In fact, Clients can also use the protocol to send a string of characters. For example, when a user replaces the contents of an entry field with a new string, the Client may then send the entire string all at once to the Server, instead of character by character.

[0117] In accordance with one embodiment of the invention the system is session-based, in that the server knows or recognizes when subsequent requests originate at the same Client. Thus, in responding to a character the Server receives from a Client it can use the history of data that has been sent to and from the current user. In one embodiment, the system stores user preferences with each Service, so that they are always available to the Client, (i.e., they are independent of the physical location of the client). Furthermore, client authentication and a billing system based on actual data and content use by Clients are supported. For faster response, the Server may predict input from the Client based on statistics and/or algorithms.

[0118] The system is bi-directional and asynchronous, in that both the Client and the Server can initiate communications at any moment in time. The functionality of the system is such that it can run in parallel with the normal operation of clients. Tasks that clients execute on the system are non-blocking, and clients may resume normal operation while the system is performing those tasks. For example, a communication initiated by the Client may be a single character that is sent to the Server, that responds by returning appropriate data. An example of a communication initiated by the Server is updating the information provided to the client. Because the system is session-based it can keep track of database information that has been sent to the Client. As information changes in the database, the Server sends an updated version of that information to the Client.

[0119] Embodiments of the system may be implemented as a multi-tier environment This makes it scalable because the individual tiers can be replicated as many times as necessary, while load balancing algorithms (including but not limited to random and round robin load-balancing) can be used to distribute the load over the copies of the tiers. One skilled in the art would appreciate that it is not necessary to replicate the tiers. Indeed, there may be only a single copy of each tier, and that all tiers (Client, Server, and Service) may be running on a single computer system.

[0120] FIG. 1 illustrates the general outline of a system that embodies the present invention. As shown in FIG. 1 there may be various Clients 101 using the system. These Clients use a communication protocol 102 to send information, including but not limited to single characters, and to receive information, including but not limited to lists of strings and corresponding metadata. At least one Server 103 receives information from the Client, and sends information to the Client. In a typical embodiment if there is a plurality of Servers, then the system can be designed so that each Client connects to only one of them, which then relays connections to other Servers, possibly using load-balancing algorithms. Servers have a communication link 104 to a Service 105, which they use to obtain the information that they send to the Client.

[0121] FIG. 2 is a schematic illustrating an embodiment of the present invention, and displays a five-tier system that has a user interface in which user interface elements use the present invention to assist the user in performing its tasks. For purposes of illustration, FIG. 2 displays just one session and one content Service. In an actual implementation there may be multiple concurrently active sessions, and there may be more than one content Service that Clients can use. As shown herein, the first of the five tiers is a Client tier 201. The Client tier contains the user interface and the Client components that are needed to use the system. The second tier is a Server or server process 206, which handles the queries that Clients execute, and in return displays results to the Client. Service 213, which corresponds to 105 of FIG. 1, is a logical entity consisting of three more tiers: a Syndicator 214, a Content Channel 219 and a Content Engine 224. The Syndicator provides access to a number of Content Channels and performs accounting services based on actual data use. The Content Channel provides a specific type of information from a specific source (i.e. the Content Engine). The Content Engine is the actual source of any content that is made available through the QuestObjects system. The Client tier 201 corresponds to the client 101 in FIG. 1. In this example, the Client may be an application (and in some embodiments a web application) with a user interface that accesses the system of the present invention. As used in the context of this disclosure a user interface element that uses the present invention is referred to as a

"Questlet." A Client can contain one or more Questlets **202** (e.g. an input field or a drop down list. **FIG. 3** described later contains three examples of such Questlets. A Questlet is always associated with at least one Client Quester **203**. Questers are objects that tie a QuestObjects input buffer (containing input from the Client) to a QuestObjects Result Set returned from a QuestObjects Server. Questers exist on both the Client and Server, in which case they are referred to as a Client Quester and a Server Quester, respectively. Every Client Quester has one corresponding Server Quester. In accordance with the invention, any event or change that happens in either one of them is automatically duplicated to the other so that their states are always equal. This synchronization mechanism is fault-tolerant so that a failure in the communication link does not prevent the Questers from performing tasks for which they do not need to communicate. For example, a Client Quester can retrieve results from the cache, even if there is no communication link to the Server. Each single Quester accesses exactly one QuestObjects Service, i.e. one specific Content Channel offered by one specific Syndicator. At initialization of the Client, the Questlet tells its Quester which Service to access. In one embodiment a Service is stored or made available on only one Server within a network of Servers. However, this is transparent to the Client because each Server will forward requests to the right computer if necessary. The Client does not need to know the exact location of the Service.

[0122] To communicate with its Server Quester **208**, each Quester in a session uses a controller **204**. The system contains at least one Client Controller **204** and a Server Controller **209**, which together implement the network communication protocol **205** of the present invention. Client Controllers may cache results received from a Server, thus eliminating the need for network traffic when results are reused.

[0123] Client Questers are managed by a Questlet, which create and destroy Questers they need. In a similar fashion, Server Questers are managed by a Session **207**. When a Client Quester is created, it registers itself with the Client Controller. The Client controller forwards this registration information as a message to the Session using the Server Controller. The Session then checks if the Persistent Quester Store **210** contains a stored Quester belonging to the current user matching the requested Service and Query Qualifier. If such a Quester exists, it is restored from the Persistent Quester Store and used as the peer of the Client Quester. Otherwise, the Session creates a new Server Quester to be used as the Client Quester's peer.

[0124] A Time Server **211** provides a single source of timing information within the system. This is necessary, because the system itself may comprise multiple independent computer systems that may be set to a different time. Using a single-time source allows, for example, the expiration time of a Result Set to be calibrated to the Time Server so that all parts of the system determine validity of its data using the same time.

[0125] Server communication link **212** is used by the Server to send requests for information to a Service, and by a Service to return requested information. Requests for information are Query objects that are sent to and interpreted by a specific Service. Query objects contain at least a string used by the Service as a criterion for information to be retrieved, in addition to a specification of row numbers to be returned to the Client. For example, two subsequent queries may request row numbers **1** through **5**, and **6** through **10**, respectively. A query object may also contain a Qualifier that is passed to the appropriate Service. This optional Qualifier contains attributes that are needed by the Service to execute the Query. Qualifier attributes may indicate a desired sort order or in the example of a thesaurus Service may contain a parameter indicating that the result list must contain broader terms of the Query string. Services use the communication link to send lists of strings (with their attributes and metadata) to Servers. Server communication link **212** is also used by Server Questers to store and retrieve user preferences from a Syndicator's Preference Manager.

[0126] Questers use Services to obtain content. A Service is one of the Content Channels managed by a Syndicator. When a Quester is initialized, it is notified by its Active Component of the Service it must use. The Service may require authentication, which is why the Syndicator provides a User Manager **215**. If a Client allows the user to set preferences for the Service (or preferences needed by the Active Component), it may store those preferences using the Syndicator's Preference Manager **216**. The Server (i.e. Server Quester) only uses the Syndicator for authentication and preferences. To obtain content, it accesses the appropriate Content Channel directly. The Content Channel uses its Syndicator to store usage data that can be later used for accounting and billing purposes. Usage data is stored in a Usage Statistics Store **217**.

[0127] Content communication link **218** is used by Content Channels to send usage data to their Syndicator, and to retrieve user information from the Syndicator. The Content Channel is a layer between the QuestObjects System, and the actual content made available to the system by a Content Engine **224**. Each Content Channel has a corresponding Query Manager **220** that specifies the type of query that can be sent to the corresponding Content Engine, and defines the types of data that can be returned by the Content Channel.

[0128] Specification of query type comprises a set of Query Patterns and Query Filters that are used by the Server Quester to validate a string before the string is sent to the Content Channel as a QuestObjects Query. For example, a query type "URL" may allow the Server Quester to check for the presence of a complete URL in the input string before the input string is sent to the Content Channel as a query. A query type "date" might check for the entry of a valid date before the query is forwarded to the Content Channel.

[0129] The Query Manager optionally defines the types of string data that can be returned to the Client by the Content Channel. Specific Active Components at the Client can use this information to connect to Services that support specific types of data. Examples of string types include: simple terms, definitional terms, relational terms, quotes, simple numbers, compound numbers, dates, URLs, e-mail addresses, preformatted phone numbers, and specified XML formatted data etc.

[0130] The Query Manager **220** retrieves database information through a Content Access Module **221**. The Content Access Module is an abstraction layer between the Query Manager and a Content Engine. It is the only part of the system that knows how to access the Content Engine that is linked to the Content Channel. In this way, Query Managers can use a standardized API to access any Content Engine. To reduce information traffic between Content Channels and Content Engines, Content Channels may access a content-based cache 222 in which information that was previously retrieved from Content Engines is cached. Engine communication link 223 is used by Content Access Modules to communicate with Content Engines. The protocol used is the native protocol of the Content Engine. For example, if the Content Engine is an SQL based database system then the protocol used may be a series of SQL commands. The Content Access Module is responsible for connecting the Content Engine to the QuestObjects System.

[0131] Content Engines 224 are the primary source of information in the system. Content Engines can be located on any physical computer system, may be replicated to allow load balancing, and may be, for example, a database, algorithm or search engine from a third-party vendor. An example of such an algorithm is Soundex developed by Knuth. Content Engines may require user authentication, which, if required, is handled by the Syndicator (through the Content Access Module).

[0132] The invention uses Content Engines as a source of strings. One skilled in the art would understand that a string may, for example, contain a URL of, or a reference to any resource, including images and movies stored on a network or local drive. Furthermore, strings may have metadata associated with them. In one embodiment, strings might have a language code, creation date, modification date, etc. An entry in a dictionary may have metadata that relates to its pronunciation, a list of meanings and possible uses, synonyms, references, etc. A thesaurus term may have a scope note, its notation, its source and its UDC coding as metadata, for example. Metadata of an encyclopedia entry may include its description, references, and links to multi-media objects such as images and movies. A product database may have a product code, category, description, price, and currency as metadata. A stock quote may have metadata such as a symbol, a company name, the time of the quote, etc. Instructions to a control system may contain parameters of those instructions as metadata. For example, the instruction to open a valve can have as metadata how far it is to be opened.

[0133] FIGS. 3A-3C contain three examples of the Questlets that can be used with the system, i.e., the User Interface Elements that access the QuestObjects system. In **FIG. 3A**, a series of representations of an auto-completing entry field are shown, such as might be used in an application window or on a web form, that accesses a single QuestObjects Service, and allows for auto-completion of, in this example, a U.S. state name. **FIGS. 3B and 3C** depict two different presentation forms of the same complex Questlet that access a number of QuestObjects Services simultaneously.

[0134] Users should be able to clearly recognize the availability of QuestObjects Services in an application. As shown in **FIG. 3A**, and particularly in the auto-complete entry field example screen element 302, clear symbols are displayed at the right end of the field. A small disclosure triangle 308 is displayed in the lower right-hand corner, and serves as an indicator to the user that a QuestObject is being used. A reserved space herein referred to as the "status area", and located above the disclosure triangle 301 is used to display information about the state of the QuestObjects system. The successive shots of this screen element 302 through 307 show some of the different kinds of states in this status area. Screen element 302 depicts an empty data field with an empty status area. The screen element 303 shows the same field immediately after the user enters a character "N". On receiving the "N"input, the Questlet immediately checks its internal entry cache for available auto-complete responses. If the cache does not contain a valid string (either because the cache is empty, because the cache is incomplete for the entry character, or because one or more cached strings have expired) the QuestObjects system sends a query to the QuestObjects Service. This sending process is indicated by a network access symbol in the status area 304 which is in this embodiment takes the form of a left and right facing arrows.

[0135] Screen element 305 shows the entry field after the Server has sent one or more auto-complete strings back to the Questlet. This example situation is typical of these instances in which the user did not enter a second character after the original "N" before the QuestObjects system responded. The QuestObjects system is inherently multi-threaded and allows the user to continue typing during access of the QuestObjects Service. The screen element status area of 305 now displays a small downward facing arrow indicating that there are more available auto-complete answers. In this case, the entry field has displayed the first one in alphabetic order.

[0136] Screen element 306 shows the same entry field after the user has hit the down arrow key or clicked on the arrow symbol in the status area. The next available auto-complete response in alphabetical order is displayed. The double up and down pointing arrows in the status area now indicate that both a previous response (in this example, "Nebraska") and a next response are available.

[0137] Screen element 307 shows the same entry field after the user has typed two additional characters, "e" and "v". As shown in this example, the status area changes to a checkmark indicating that there is now only one available auto-complete match for the characters entered. The user can at any point use the backspace key on their keyboard (or perform other actions defined in the Questlet) to select different states, or can leave the entry field to confirm his selection. At this time, the system may do several things. It can automatically accept the string "Nevada" and allow the user to move on with the rest of the entry form; or if it has been configured such it may decide to replace the string "Nevada" by the two-character state code. The QuestObjects Service not only returns strings, but also any corresponding metadata. This example of an auto-complete entry field Questlet is based on showing the response string, but other Questlets (and even invisible Active Components) may perform an action invisible to the user. In addition, a response sent to one Questlet can trigger a response in other Questlets that have a pre-defined dependency to that Questlet. For example, entering a city into one Questlet can trigger another Questlet to display the corresponding state. It will be evident to one skilled in the art, that although left, right, up and down arrows are used to indicate usually the status of the QuestObject field, other mechanisms of showing the status within the scope and spirit of the invention.

[0138] Interdependent data (which in the context of this disclosure is that data originating from a multitude of

QuestObjects Services) can be combined into a complex Questlet. Examples **309** shown in **FIG.** 3B and example **313** shown in **FIG. 3C** show a complex user interface element (Questlet) that makes multiple QuestObjects Services available to the user. In both examples the upper part of the Questlet is an entry field that may offer the auto-complete functionality described in **FIG.** 3A. By clicking on the disclosure triangle **308** shown in the earlier **FIG. 3A** (or by another action), the user can disclose the rest of the Questlet, which in this example comprises two functional areas **311** and 312. In this example, the user interface allows the user to choose a vertical presentation mode **309**, shown in **FIG. 3B** or a horizontal presentation mode 313, shown in **FIG. 3C** for the Questlet. A close box **310** replaces the disclosure triangle in the entry field, allowing the user to close areas **311** and 312. In **FIG. 3C** Area **314** shows a certain QuestObjects Service, in this case a list of "Recent Terms" accessed by the user. This Questlet allows the user to select a different QuestObjects Service for area **314** by selecting it from a popup list 319. In this example, an appropriate second Service might be "Alphabetic Listing".

[0139] In both examples of **FIGS. 3B and 3C**, area **312** displays a QuestObjects "Thesaurus Service" (Thesa) that has been selected. Additionally, in **FIG. 3C** areas **315** through **318** display four different Questers that take their data from a QuestObjects Thesaurus Service. These Questers all access the same Thesaurus and all have a dependency on the selected string in the main list of area **314**. Once the user clicks on a string in area **314** the thesaurus lists **315** through **318** are automatically updated to show the corresponding "Used For terms" UF, "Broader Terms" BT, "Narrower Terms" NT, and "Related Terms" RT from the Thesaurus Service. Questers **315** through **318** thus have a different Qualifier that is used to access the same QuestObjects Service. It will be evident to those skilled in the art that this example is not intended to be a complete description of features that a thesaurus browser (or any other Service) provides. Most thesauri offer a multitude of term relationships and qualifiers. A Questlet or part of a Questlet may provide access to a multitude of QuestObjects Services. A possible way to do this is to show multiple tabbed panes accessible through tab buttons named after the Services they represent **320**.

[0140] Data from the QuestObjects Services can be displayed by a Questlet in many forms. Thesaurus browser Questlets generally display interactive lists of related terms. Questlets can also allow users to lookup data in a reference database (dictionary, encyclopedia, product catalog, Yellow Pages, etc.) made available as a QuestObjects Service. Furthermore, Questlets can access QuestObjects Services that provide a standardized interface to search engines. These search engines may be Internet-based or can be built into existing database servers. Questlets can also access pre-defined functions made available as QuestObjects Services (such as a bank number check, credit card validation Service or encryption/decryption Service). Questlets can even access translation Services allowing on-the-fly translation of entry data. In some embodiments Questlets can retrieve multi-media data formats by receiving a URL or pointer to multi-media files or streaming media from a QuestObjects Service. In other embodiments Questlets can be used to display current stock quotes, news flashes, advertisements, Internet banners, or data from any other real-time data push Service. Questlets can provide an auto-complete or validity checking mechanism on the data present in specific fields or combinations of fields in relational database tables.

[0141] As described above, Questlets are well suited to represent QuestObjects data visually. However, a QuestObjects Client system can also contain non-visual Active Components, such as function calls from within a procedure in a program to access a QuestObjects Service. A program that needs to display a static or unchanging list of strings can use a Quester in its initialization procedure to retrieve that list from a QuestObjects Server. By calling a Quester, a stored procedure in a database can make a QuestObjects Service available to any database application. By encapsulating a Quester into an object supplied with a programming language, a QuestObjects Service can be made available to its developers. Another example of how QuestObjects Services may be accessed is through a popup menu that a user can access by clicking on a word, phrase or sentence in a document. The popup menu can include one or more QuestObjects Services by calling one or more Questers. In an application that is controlled by speech, a sound conversion engine that translates speech input into phonemes can be used to send these phonemes to a QuestObjects speech recognition Service through a Quester. As yet another example, a control system can use a Quester to send sensor readings to a Server, which then queries a special purpose content engine to return actions that the control system must perform given the sensor readings.

[0142] **FIG. 4** shows a simplified event life cycle illustrating what happens in a QuestObjects system using an auto-complete Service. The protocol of the present invention is implemented in the Client Controller and the Server Controller **400**. In an initial phase an Active Component on the Client tells its Quester to start or initialize **401** a corresponding Client Session on the current QuestObjects Server by sending a Register message to its Client Controller. The Server Controller starts a Client Session if it has not been started already. For simplicity the event trace of **FIG. 4** does not show typical error handling that normally occurs, for instance when a Session cannot be started. If the Quester was used before in the same Active Component and application, the Session may restore the Quester from a Persistent Quester Store, which may even cause a Query to be triggered immediately if the Result Set in the Quester is out of date.

[0143] The Server Quester looks up the Service in the Server's list of known QuestObjects Services, which may or may not be located on the same computer. Once the Service is found, the Client is registered and optionally authenticated by the Service. At this time, the Service **402** returns information to the Server Controller at which time the Client receives a confirmation that it was registered successfully. The Active Component can now start using the Quester it has just initialized. If the Active Component has a user interface (i.e. it is a Questlet) then it will now allow the user to start entering characters or cause other user events.

[0144] The next step in the process is to capture user input. As shown in **FIG. 4**, at point **403** a character event is generated to indicate the user has typed a character 'a' into the Questlet. The Quester sends a message to its Client Controller telling it that character 'a' must be appended to the input buffer (it will be evident to one skilled in the art that if the cursor is not at the end of the input string, typing 'a' would, for example, generate a different event to insert the character instead of append it). The Client Controller uses the protocol to synchronize the input buffer in the Server Quester by communicating to the Server Controller. The Server Controller may look up query 'a' in its Result Set cache, in which case it can return a previous Result Set to the Client without accessing the Service. Also, depending on any rules specified by the Service (as specified by a list of Query Patterns and Query Filters defined in the Query Manager of the Content Channel) and depending on the time interval between input buffer changes, the Server Quester may decide not to immediately send the (perhaps incomplete) string to the Service, as shown here.

[0145] An additional character event **404** is generated when the user has typed a second character 'b' into the Questlet. As before, a corresponding event arrives at the Server Quester. In this case, the Server Quester may deduct that the input string represents a valid query and send the appropriate query message 'ab' to the Service. After receiving a query, the Service executes it by accessing its Content Engine through the Content Access Module unless the Query Manager was able to lookup the same Query with a Result Set in the Content-based Cache. After an appropriate Result Set **405** is retrieved, the Service will return it to the Client. In some embodiments, a large Result Set may be returned to the Client in small batches. In other embodiments an incomplete Result Set may also be returned if the Content Engine takes a long time to come up with a batch of results. A QuestObjects Service may automatically 'push' updated information matching the previous query to the Client as it becomes available. A Query can also be set to auto-repeat itself **406** if necessary or desired.

[0146] At step **407** the user types a third character 'c' into the Questlet. While this character is being sent to the Server, a second and possibly third result set from the previous query is on its way to the Client. When the Client Controller decides **408** that the received Result Set 'ab' no longer matches the current input string 'abc', the second update of 'ab' is not transmitted to the Active Component. Depending on the sort order and sort attributes of the Result Set, the Client Controller may still send the second and third Result Sets to the Active Component if the second query 'abc' matches the first string of the Result Set for the first query 'ab'**409**. In that case, the user typed a character that matched the third character in the second or third Result Set, thus validating the Result Sets for the second query. Eventually the Server Quester receives notice of the third character appended to the input buffer, and sends a new query 'abc' to the Service. The Server Quester will stop the 'repeating' of query 'ab' and the Service will now execute **410** the new query 'abc' at the Content Engine, or retrieve it from the Content-based Cache.

[0147] FIG. 5 depicts a flowchart illustrating the interface between an Active Component and the present invention. As shown therein a Client Quester is initialized (step **501**) in which each active component is associated with one or more Client Questers. A loop is then entered that exits when the Active Component is destroyed (step **502**). In the loop, events are sent to the Client Quester (step **503**), such as keyboard events, click events and focus events (i.e. events that tell the system which user interface element currently has input focus). When events are sent to the Client Quester, they may result in return events from the Client Quester, such as events informing that the Result Set of the Client Quester has changed. Those events are received by the event receiver (step **504**). The event receiver waits for events from the Client Quester (step **506**) and—if events have been received (**507**)—processes them (step **508**). It will be evident to one skilled in the art that the Active Component can be multi-threaded, in that the event receiver can work concurrently with the rest of the Active Component. The Active Component may also use a cooperative multi-threading scheme where it actively handles client events and server responses in a continuous loop.

[0148] FIG. 6 shows a flow chart illustrating the Client side of the present invention. First, the Client Quester registers itself with the Client Controller (step **601**). It then enters a loop that exits when the Client Quester is destroyed (step **602**). When that happens, the Client Quester deregisters itself from the Client Controller (step **603**). During the loop the Client Quester handles events from the Active Component it belongs to. First, it waits for an event and receives it (step **604**). Then the type of the event is checked (step **605**). If it is not a character event, it is handled depending on the type and content of the event (step **606**). An example of a non-character event is a double-click on the input string, the click of a button that clears the input buffer, the addition of characters to the input buffer by a paste-action etc. If the event is a character event, the input buffer is updated accordingly and Client Questers that have dependencies with the input buffer or the Result Set also are notified (step **607**).

[0149] The next step is to get results based on the new input buffer. First, the Client Quester checks if the results are present in the client-side cache, which usually is a fast short-term in-memory buffer (step **608**); if so, they are retrieved from the cache (step **609**) and the Active Component is notified of the results (step **610**). If the results are not found in the cache, the Client Quester uses the Client Controller to send the new input buffer to the Server Quester, so that a new query can be executed (step **611**). To support this, the protocol of the present invention provides a number of messages that allow the Client Quester to send just the changes to the input buffer, instead of sending the entire input buffer. These messages include but are not limited to: inputBufferAppend, inputBufferDeleteCharAt, inputBuffer-InsertCharAt, inputBufferSetCharAt, inputBufferSetLength, and inputBufferDelete. After thus updating the Server Quester's input buffer, the Client Quester activates the result retriever to wait for new results and process them (step**612**).

[0150] The Client Quester is intended to be multi-threaded, so that it can continue providing its services to its Active Component while it waits for results from the QuestObjects Server. Therefore, the Result Retriever can be implemented to run in a separate thread of execution. In this embodiment the Result Retriever waits for results from the Server Quester (step **613**). If results have been received (step **614**), it checks whether they are usable (step **615**). Results are usable if they correspond to the latest query. If results are from a previous query (which can occur because the system is multi-threaded and multi-tier), they may also still be usable if the Client Quester can filter them to match the new input buffer (this depends on the sort flags in the Result Set). If results are usable, the Active Component is notified of the new results. This notification is also sent to other Client

Questers that have dependencies on the originating Client Quester (step 616). Received results are stored in the client-side cache, regardless of whether they were found to be usable (step 617).

[0151] FIG. 7 is a flow chart illustrating the Server side of the present invention. The first thing a Server Quester does when it is created, is to check whether its attributes can be restored from the Persistent Quester Store (step 701), based on the parameters with which it is created. If the attributes can be restored, they are restored and registered with its corresponding Service (step 702). In accordance with one embodiment, one of the restored attributes is a Result Set attribute; the Server Quester checks whether it is still up to date (step 703). If not, a query is sent to the corresponding Service if it is a pushing service or if the Query was originally set to be auto-repeating (step 704) and (in a separate thread of execution) the Server Quester waits for the results of that query and processes them (step 705).

[0152] If the Server Quester's attributes could not be restored, it initializes itself and registers itself with the correct service which is one of the initialization parameters (step 706). If the Client Quester was created with a default input buffer, the Server Quester may automatically send the corresponding Query to the Service. At this point, the initialization process is complete and the Server Quester enters a loop that exits when the Quester is destroyed (step 707). During the loop, the Server Quester checks whether the Query String is valid, using the validation attributes of the Service (Query Pattern and Query Filter) (step 708). If the query is valid, the Server Quester checks if the server-side cache has the results for the Query String (step 709). If not, a new Query is sent to the Service (step 710). After that, the results are retrieved (either from cache or from the Service) and processed (step 711).

[0153] After validating (and possibly processing) the Query String, the Server Quester waits for messages from the Client Quester notifying of changes to the input buffer (step 712). If such a message is received, the input buffer is updated accordingly (step 713), and the loop is re-entered (step 708).

[0154] The processing of query results is performed in a separate thread of execution. The process performed in this thread starts by obtaining the Result Set (step 714), either from the server-side cache or from the Service depending on the result of the decision in step 709. When these results are obtained (step 715), they are sent to the Client Quester (step 716) either as part of the Result Set or as the entire Result Set, depending on parameters set by the Client Quester and are stored in the server-side cache (step 717). In addition, the Service is notified of actual results that have been sent to the client (step 718). If the results were pushed by the Service (step 719), this thread starts waiting for new results to be processed; otherwise, the thread stops.

[0155] FIGS. 8A-8D illustrate and object model of an embodiment of the present invention. FIG. 8A illustrates the base portion of the model containing the entities that are not specific to either QuestObjects Clients, QuestObjects Servers, or QuestObjects Services. FIG. 8B displays the entities that are specific to the QuestObjects client. FIG. 8C contains the entities specific to the QuestObjects Server. FIG. 8D shows the entities specific to the QuestObjects Service.

[0156] Each of FIGS. 8A through 8D show object models of one particular embodiment of the present invention, using UML (Unified Modelling Language) notation. Note that in the figures some of the entities have a name that starts with one of the words 'base', 'client', 'server', and 'service', followed by two colons. Those entities are merely references to entities in the subfigure indicated by the word before the two colons. For example, the entity named 'service::QoService' in FIG. 8A is a reference to the 'QoService' entity in the figure of the service part, namely FIG. 8D. It will be evident to one skilled in the art that the model shown is purely an illustrative example of one embodiment of the invention and that other models and implementations may be developed to practice the invention while remaining within the spirit and scope of the this disclosure.

[0157] The base part of the system—depicted in FIG. 8A—comprises entities that are not specific to one of the tiers of the QuestObjects system. One of the most important entities shown in FIG. 8A is QoString, the QuestObjects String. QoString models the strings that the QuestObjects System handles. A QoString has at least a value, which is the sequence of (Unicode) characters itself. To guarantee a minimum performance level, i.e. one in which the communication takes as little time as possible, this value has a limited length (e.g. of 256 characters). Furthermore, a QoString may have a key and metadata. The key (if any is present) is the identifier (i.e. the primary key) of the QuestObjects String in the database from which it originates. This key can be used to retrieve data from the database that is related to the QuestObjects String. Metadata of a QoString can be any additional data that is provided with the QoString's value. Metadata of a QoString is XML formatted and has a limited length (e.g. 2048 bytes), in order to ensure that QoStrings can be exchanged between the tiers of the QuestObjects System without compromising efficiency. If the QoString originates from a Content Channel, it may also have a fetch Time, namely the timestamp of when the QoString was retrieved from the underlying content provider. It also may have an expiration Time indicating how long the data in the QoString is to be considered valid. Optionally a QoString can have a type, which is a reference to a QoType object. (Note that for maximum efficiency the types are not actually stored in the QoStrings, because it is very likely that many QoStrings in a QoResultSet have the same type. Storing the types in the strings would unnecessarily increase network traffic.)

[0158] The QoType object models the concept of a string's type. It has a string typeString that contains the description of the type and an indicator typeIndicator that defines the meaning of the description (typeString). Examples of string types are: the DTD or Schema of the string's value in these cases in which it is XML formatted (or, alternatively, the URL of the DTD or Schema), the number formatter in the case it is a number, and the date (and/or time) formatter in the case it is a date (and/or time). Table 1 shows an example of the use of types, especially type indicators.

TABLE 1

Value of typeIndicator	Meaning of typeString
0	typeString contains the name of the type
64	typeString contains a string formatter
65	typeString contains a number formatter
66	typeString contains a date formatter

MO000031

TABLE 1-continued

Value of typeIndicator	Meaning of typeString
128	typeString contains a DTD
129	typeString contains a Schema
160	typeString contains the URL of a DTD
161	typeString contains the URL of a Schema
255	custom type; typeString is the type's name

[0159] In the example shown in Table 1, bit 7 of the typeIndicator is on if typeString is XML related, bit 6 is on if typeString is some formatter, and bit 5 is on when typeString is a URL. This name must follow the same naming scheme as Java packages: They must use the Internet domain name of the one who defined the type, with its elements reversed. For example, custom types defined by MasterObjects would begin with "com.masterobjects.".

[0160] The QoQuery entity models the specification of a QuestObjects Query. It includes a queryString that contains the value the Content Channel is queried for (which is named queryString in the figure). In addition to the queryString, QoQuery has a property 'qualifier' that can hold any other attributes of the query. The format and meaning of the qualifier's contents is defined by the Content Channel that executes the query. Furthermore, it can be specified which row numbers of the total result set must be returned using the property 'rownums'. The property 'requestedTypes' can optionally hold a list of QoTypes, limiting the types of the strings that will result from the query. The 'timeout' property can be used to specify a maximum amount of time execution of the query may take.

[0161] Queries may include a type (QoQuerytype). Query types are similar to QoType (i.e. String Types), and can be used by QuestObjects Clients to find all QuestObjects Services that support a certain kind of Query.

[0162] The result of a query is represented by the QoResultSet entity. QuestObjects Result Sets are collections of QuestObjects Strings that are sent from a QuestObjects Server to a QuestObjects Client in response to a query. QoResultSets are created and filled by a QuestObjects Service (to which QoResultSet has a reference named 'service'), based on the QoQuery to which the QoResultSet has a reference. Actual results are stored as an array of QoStrings in the 'strings' property. Elements of the QuestObjects Result Set (i.e. QoStrings) may be selected, as indicated by the 'selected' property that is a list of indices in the strings array of selected strings. Also, one of the QoStrings may be marked as current as indicated by the 'current' property. (When a QoString is marked as current it means that all operations are performed on that QoString, unless another one is explicitly specified.) QuestObjects Result Sets include an attribute 'ordered' that indicates whether the QoStrings in the QoResultSet are ordered. Sometimes, especially when a QuestObjects Result Set is narrowed down by a new Query, the fact that the QoResultSet is ordered may mean that the QuestObjects Client does not need to actually execute a new Query; instead, it can filter the previous QuestObjects Result Set itself according to the new queryString.

[0163] As further described below, Server Questers may have a QuestObjects Result Set, of which only a part is sent to the QuestObjects Client. At all times, the 'rownums' property of QoResultSet indicates the row numbers of QoStrings that are actually present in the QoResultSet. The rownums property may have different values for corresponding QoResultSets on the QuestObjects Server and the QuestObjects Client. The same holds for the 'strings' property. The 'complete' property is the percentage of the QoStrings in the server-side QoResultSet that is present in the corresponding client-side QoResultSet as well. The property 'totalNumberOfStrings' indicates the total number of QoStrings in the QoResultSet, whether actually present or not. For server-side QoResultSets this number is always equal to the length of the 'strings' array, but for client-side QoResultSets the number may be smaller.

[0164] Finally, result sets include an identifier 'resultSetId'. Every time a Client Quester uses the protocol of the present invention to send something to the Server Quester that may result in a new QuestObjects Result Set, it includes a request identifier. This identifier is then copied in the resultSetId when the QuestObjects Result Set is sent to the Client Quester. In this way Client Questers know which request the QuestObjects Result Set belongs to. (This is important because the system is asynchronous and on occasions it may occur that a newer QuestObjects Result Set is sent to the client before an older one. The request identifier and QuestObjects Result Set identifier allow the Client Quester to detect and handle this.)

[0165] The core entity in the figure is QoQuester. QoQuester is the superclass of both QoClientQuester (part of the client and thus depicted in **FIG. 8B**) and QoServerQuester (depicted in **FIG. 8C**). The QoQuester entity models the Quester concept. Its primary task is to maintain an input buffer, to make sure that QuestObjects Queries are executed and to store and provide access to the QuestObjects Result Sets returned by QuestObjects Services in reply to QuestObjects Queries. At all times, a QoQuester holds one QoResultSet that contains the results of the latest QuestObjects Query. (Note that a QoQuester may hold other QoResultsSets as well, for example for optimization purposes.) Client Questers and Server Questers exist in a one-to-one relationship with each other: for every Client Quester there is exactly one corresponding Server Quester, and vice versa. All properties listed in QoQuester are present and equal, both in the Client Quester and in the corresponding Server Quester. An important exception is the resultSet property. In the Server Quester, this is always the entire QuestObjects Result Set of the latest Query. However, in order to minimize network traffic the Server Quester is intelligent about the portion it actually sends to the Client Quester. Questers include a property 'minimumBatchTime' that indicates the minimum amount of time that should pass before the Server Quester sends results to the Client Quester. This allows the Server Quester to accumulate results and send them as a single action instead of as a separate action for each result. There are two situations in which the Server Quester may ignore this minimum batch time:

[0166] (a) when the result set is complete before the minimum batch time has passed, and

[0167] (b) when the number of accumulated results exceeds the number indicated by the 'resultSetBatchSize' property before the minimum batch time has passed.

[0168] If, for whatever reason, the Server Quester postpones sending the accumulated results to the Client Quester, the (optional) 'maximumBatchTime' property indicates how long it may postpone the sending. Even if no results are available yet, when the maximumBatchTime passes, the Server Quester must notify the Client Quester thereof.

[0169] Results are sent to the Client Quester in batches, the size of which is indicated by the 'resultSetBatchSize' property. Occasionally, the Server Quester may deviate from this batch size, notably when the number of results that is not present on the client is smaller than the batch size or when the maximumBatchTime has passed. This concept can be taken even further, for example when the batch size is 10 results and the Server Quester has 11 results, the Server Quester may send them all, even though it exceeds the batch size, because sending one extra result with the other 10 is probably more efficient than sending a single result in a separate batch at a later point. The Server Quester can use the 'clientMaximumLatency' to make such decisions; it indicates the maximum expected amount of time that elapses between sending a message and receiving its response. The higher this value, the more likely it is that sending the eleventh result with the other ten is more efficient.

[0170] Questers include an input buffer. The content of the input buffer is what the QuestObjects Service will be queried for. In the Client Quester, the input buffer is controlled by the application that uses the QuestObjects system. For example, an application with a graphical user interface may update the input buffer according to key presses in one of its input fields. The Client Quester keeps the input buffer of its corresponding Server Quester up to date using the protocol of the present invention.

[0171] Properties 'highestReceivedResultSetId' and 'latestRequestId' are used to detect when QuestObjects Result Sets are received out of order. As with the 'resultSetId' property of the QoResultSet, every QuestObjects Result Set includes an identifier. The 'highestReceivedResultSetId' property stores the highest of all received QuestObjects Result Set identifiers. If a Client Quester only needs the latest results, it can simply discard received QuestObjects Result Sets that have a lower identifier than 'highestReceivedResultSetId'. The 'latestRequestId' is the identifier of the latest request. The QuestObjects Result Set with an identifier that matches 'latestRequestId' holds the results of the latest request.

[0172] The remaining properties of QoQuester store the QuestObjects Service the Quester uses ('service'), the optional qualifier that Queries to this QuestObjects Service need ('qualifier'), the types the Quester can handle ('types'), whether an application proxy is needed, and the optional function of the Quester in the application ('applicationFunction', used by the application proxy mechanism to determine how the value of the Quester is to be passed to the application/web server). In addition, if the update interval property 'autoUpdateInterval' is set to a non-zero value, the Server Quester will automatically repeat the last Query with that interval. This is useful for QuestObjects Services that are not capable of pushing results themselves. A mechanism is required to allow any other entity to be notified of changes in the Quester. There are many ways this can be done. As an example in the embodiment shown in FIGS. 8A-8D an event mechanism is included that involves event listeners and event handlers, very similar to the Java2 event mechanism. An entity that wants to be notified of changes must implement the QoQuesterChangeListener interface and then be added to the Quester's 'changeListeners' property (using the method 'addQuesterChangeListener'). When the Quester changes, it will call the 'questerChanged' method of all registered QoQuesterChangeListeners with a QoQuesterChangeEvent as a parameter. The QoQuesterChangeEvent holds a description of the changes of the Quester; it has a reference to the Quester that raised the event and an event type. In **FIG. 8** three event types are displayed (INPUT_BUFFER_CHANGED indicates that the Quester's input buffer has changed, RESULT_SET_CURRENT_CHANGED indicates that the current item of the Quester's Result Set has changed, and RESULT_SET_SELECTED_CHANGED indicates that the list of selected results in the Quester's Result Set has changed). More event types can be added as desired.

[0173] Another important entity in **FIG. 8A** is QoController. QoController is the entity that implements the protocol of the present invention. In addition, it knows how to buffer usage statistics and also handles the caching of result sets. QoController includes two subclasses (QoClientController and QoServerController), depicted in **FIG. 8***b* and **FIG. 8***c*, respectively. Buffering of usage statistics is an optimization that eliminates the need of exchanging usage data between the layers of the system every time a result is used. Instead, the QuestObjects Controller buffers that data and flushes the buffer when the statisticsBufferFlushTime has passed. Caching is an optimization as well. Caching is done by the QoResultsCache entry, to which the QuestObjects Controller has a reference. The QoResultsCache has a list of cached entries ('resultsCacheEntries'). The entry of the cache is modeled as QoResultsCacheEntry, an entity that has a list of QuestObjects Result Sets for combinations of query strings and qualifiers (as defined in QoQuery).

[0174] The last entity in **FIG. 8A** is QoQueryValidator. QoQueryValidator is an abstract class that defines the method 'isValid'. This method has a query string as a parameter and returns either 'true' or 'false'. QuestObjects Services may declare and publish a QoQueryValidator. By doing so, they allow the QuestObjects Server to verify the validity of a query string without actually having to send it to the QuestObjects Service, thus eliminating network traffic for invalid query strings.

[0175] **FIG. 8B** displays the minimal entities every QuestObjects Client must have. Every client of the QuestObjects System at least has a Client Controller QoClientController. QoClientController is a subclass of QoController that implements the client side of the protocol of the invention. Applications using the QuestObjects System do so through Client Questers, modeled as QoClientQuester. QoClientQuester is the subclass of QoQuester that implements client-specific Quester functionality. The figure contains the entity 'ActiveComponent'. It represents some entity that uses the QuestObjects System through one or more Client Questers.

[0176] **FIG. 8C** shows the server part of the embodiment of the present invention, and includes the QoServerQontroller, one of the subclasses of QoController. QoServerController implements the server-side part of the protocol of the present invention. In addition, it maintains a list of sessions running on the server, and it has references to a Persistent Quester Store, an optional Service Directory and a list of optional Application Host Synchronizers. For security reasons, one implementation of the QuestObjects System may require that only certified clients can connect to the system. A boolean 'requiresCertification' indicates this.

[0177] The QuestObjects System is session-based. This means that clients that use the system are assigned to a session, modeled by the QoSession entity. Every session has a unique identifier, the 'sessionid'. The QoSession entity maintains a list of Server Questers that are active in the session (stored in the 'serverQuesters' property). Furthermore, it has a reference to the Server Controller through which a QuestObjects Client is using the session.

[0178] QoServerQuester is the server-side subclass of QoQuester. It includes a reference to the session it is being used in (the 'session' property). Furthermore, when the QuestObjects Service that the Quester uses has a Query Validator, QoServerQuester has (a reference to) a copy of that Query Validator, so that query strings can be validated before they are actually sent to the QuestObjects Service. The QoPersistentQuesterStore is an entity that is able to store a user's session and to restore it at some other time, even when the session would normally have expired or even when the same user is connecting from a different client machine. To this end, QoServerQuester has two methods 'store' and 'restore'. The first, 'store', returns a QoStoredQuester, which is a (persistent) placeholder of the Server Quester that contains all relevant data of that Server Quester. The second, 'restore', needs a QoStoredQuester as an argument. The two are each other's inverse, which means calling 'store' on a QoServerQuester and then calling 'restore' on the result creates a new QoServerQuester that is an exact copy of the original QoServerQuester.

[0179] QoServiceDirectory acts as a Yellow Pages or directory of QuestObjects Services. For each QuestObjects Service it stores the name and address, as well as the address of the QuestObjects Server through which the Service can be accessed. Furthermore, Services' profiles are additionally stored to allow clients to find all QuestObjects Services satisfying desired criteria.

[0180] Finally, QoAppHostSynchronizer is the AppHost Synchronizer. QoAppHostSynchronizer has its address as a property ('appHostAddress').

[0181] FIG. 8D depicts the service part of the embodiment of the present invention. Content is disclosed through Content Channels (the QoContentChannel entity). Content Channels use Content Access Modules (QoContentAccessModule) to obtain their data in a standardized way, so only the Content Access Module knows how to communicate with the underlying data source. Content Channels are organized in Syndicators (the QoSyndicator entity), and each syndicator includes a list of Content Channels. Each Quester in the QuestObjects System uses a specific Content Channel of a specific Syndicator. This is called a QuestObjects Service, namely one of the Content Channels of a Syndicator. The property 'subscriptionRequired' indicates whether the user needs to be a registered user to be allowed to use the Service. If it is false, only users listed in 'users' may use the Service. Users can be subscribed to QuestObjects Services, which is modeled by the QoSubscription entity. Statistics are kept per Content Channel using the QoUsageStatisticsStore entity. Content Engines optionally have a Query Validator that the QuestObjects Server may use to validate Query Strings before sending them off to the QuestObjects Service. In addition, Content Channels have a profile that consists of a Content Channel's description, a list of types (QoType) of QuestObjects Strings the Content Channel can provide, an optional list of DTDs of that metadata of QuestObjects Strings from the Channel conforms to, and an optional list of Query Types the Content Channel accepts.

[0182] QuestObjects Servers communicate with QuestObjects Services through the QoServiceSession. The QoServiceSession has a static reference to the QuestObjects Service it belongs to, as well as a static array of IP addresses of QuestObjects Servers that are allowed to connect to the QuestObjects Service. In some versions of the QoServiceSession the array of IP addresses can be replaced by a list of addresses and netmasks, or by IP address ranges. Every instance of QoServiceSession has the IP address of the server that is using the session ('serverAddress'), a connection Timeout indicating the maximum period of idle time before the Service Session is automatically ended, and a serviceSessionId that can be used to refer to the Service Session.

[0183] As described above, a QuestObjects Service is one of the Content Channels of a Syndicator, so QoService has a reference to both ('syndicator' and 'contentChannel'). The property 'listable' indicates whether the Service may be listed in a Service Directory (server::QoServiceDirectory). If not, the Service can only be used if the application writer (i.e. the programmer using the QuestObjects to develop an application) knows that it exists and where it is available. The property 'name' is the Service's name, used in the Service Directory amongst others. This name must use the same naming scheme as the names of custom types. The boolean 'subscriptionRequired' indicates whether users must be subscribed (modeled by QoSubscription) to the Service in order to be allowed to use it. If the Content Engine of this Service's Content Channel requires login, 'contentEngineLoginName' and 'contentEngineLoginPassword' are the name and password with which is logged in. Finally, 'pricingInfo' contains information about the costs involved in using the Service. It is formatted as XML, conforming to a well-defined structure (i.e. DTD or Schema).

[0184] A Content Channel has a name (the 'name' property) and a profile (QoContentChannelProfile). The profile provides information about the Content Channel, namely about the Query Types it accepts ('queryTypes'), the types of the Strings it can provide ('types'), and the DTDs that QuestObjects Strings' metadata conforms to. In addition, it has a textual 'description' of the content the Content Channel discloses.

[0185] Content Channels also have properties that define the criteria Query Strings have to satisfy. The property 'queryStringMinLength' defined the minimum length a valid query has. Alternatively or additionally, 'queryStringRegularExpressions' may contain a list of regular expression describing valid Query Strings (meaning that Query Strings have to match at least one of the regular expressions). The property 'queryStringFilters' may hold a list of regular expressions and replacement strings that can transform Query Strings in a well-defined manner (for example the way the standard Unix utility 'sed' does it). Instead of using these three properties, Content Channels may define a QoQueryValidator (described above in **FIG. 8A**). If there is a Query Validator, 'queryStringMinLength', 'queryStringRegularExpressions', and 'queryStringFilters' are ignored.

[0186] As described above, Syndicators may have a list of users. Users (QoUser) have a name and a password, as well as a list of subscriptions (QoSubscription). QoSubscription models a user's subscription to a Service (the 'service' property). The properties 'startDate' and 'expirationDate' define the time frame during which the subscription is valid. Outside that time frame the user will be denied access through the subscription. The maximum number of queries the user may run in the Service is stored in the 'queryLimit' attribute. The 'queryLimitReset' defines when the query counter is reset. For example, if query limit is 10 and queryLimitReset is 7 days, the user may run 10 queries per week. (If query limit equals zero the number of queries is unlimited and queryLimitReset is ignored.) The property 'resultLimit' stores the maximum number of results the user may receive from the subscription. Similar to 'queryLimitReset', 'resultLimitReset' defines how often the result counter is reset. If 'resultLimit' equals zero the number of results is unlimited and 'resultLimitReset' is ignored. The property 'pushAllowed' indicates whether the user may use the Service in pushing mode. If so, 'pushIntervalLimit' indicates the minimum amount of time that has to pass between two pushes. A 'historyAllowed' variable indicates whether a history is kept of the use of the subscription; if so, 'historyLimit' indicates the maximum size of the history. If the maximum size is exceeded, the oldest history data is deleted so that the size of the history is below the maximum size again. If 'historyLimit' equals zero, the size of the history is unlimited. Finally, a 'usageAnonymous' variable indicates that the QoUsageRecords that are generated for this subscription must not contain user information (this is necessary because of privacy issues).

[0187] If 'keepServiceStatistics' is true, then the QoUsageStatisticsStore can store three kinds of statistics:

[0188] statistics about Strings that have been displayed on the client; the 'keepClientDisplayedStatistics' indicates whether this kind of statistics are kept.

[0189] statistics about Strings that have actually been selected on the client; the 'keepClientSelectedStatistics' indicates whether this kind of statistics are kept.

[0190] statistics about Strings that have a used on the client; the 'keepClientUsedStatistics' indicates whether this kind of statistics are kept.

[0191] The Client Quester determines the exact meaning of the three kinds of statistics. In the case of web applications, a string is generally considered displayed when the Client Quester accesses it in its QuestObjects Result Set. It is considered selected when a new Query is executed with the String as Query String. It is considered used when the form on which the Client Quester is active is submitted with that String. The actual data is stored as a list of QoUsageRecords in the propery 'records'.

[0192] A QoUsageRecord holds usage information about a QuestObjects String or a number of QuestObjects Strings. If, in one Service Session, a Quester gets the same Result Set more than once (consecutively), the usage data of each of the Strings in the Result Set is grouped in one QoUsageRecord. However, if 'stringKey', 'stringValue', 'rowInResultSet', or 'totalRowsinResultSet' changes, a new QoUsageRecord must be used from that point on. The properties of QoUsageRecord mean the following:

[0193] stringKey: if available, this is the unique key of the QuestObjects String as provided by the Content AccessModule.

[0194] stringvalue: the value of the QuestObjects String.

[0195] rowInResultSet: the row of the QuestObjects String in its QuestObjects Result Set.

[0196] totalRowsInResultSet: the number of rows the QuestObjects String's Result Set had.

[0197] dateReturnFirst: the timestamp of the first time the QuestObjects String was returned by the Content Channel.

[0198] dateReturnLast: if the QoUsageRecord represents a group of usage events, this is the timestamp of the last event.

[0199] clientDisplayed: indicates whether the QuestObjects Client that received the QuestObjects String considers it to be displayed.

[0200] clientSelected: indicates whether the QuestObjects Client that received the QuestObjects String considers it to be selected.

[0201] clientUsed: indicates whether the QuestObjects Client that received the QuestObjects String considers it to be used.

[0202] applicationName: the name of the application to which the Quester that received the QuestObjects String belongs.

[0203] appliationFunction: the function (if available) of the Quester that received the QuestObjects String.

[0204] activeComponentId: the identifier of the Active Component that received the QuestObjects String.

[0205] user: the identifier of the user that saw/selected/used the String. If the user's subscription has 'false' as value of 'usageAnonymous', then this property is empty.

[0206] Queries are executed by QoQueryExecutors. A Query Executor has a reference to the Service Session in which the Query is executed, it has a reference to the Query itself, and it also has a reference to the Server Quester that has the Query executed. This reference may be a remote object when Corba is being used, for example. If some proprietary protocol is used, it may just be the unique identifier of the Server Quester.

[0207] **FIG. 9** shows a method for using the present invention in systems that have limited technical capabilities on the Client side, such as, for example, web browsers with embedded Java applets. If developers of client systems have not integrated Client components of the present invention into their client software, then Client components needed for the present invention must be present as Plug-ins, DLL's, or an equivalent device, or they must be downloaded to the client computer as applets. These applets can be written in the Java language, when they are needed. For security reasons, such Client systems including web browsers usually do not allow 'foreign' software (i.e. software that is not an integral part of the web browser) to influence or change data entered by the user before it is sent to the application server (in this case the web server). Without an additional infrastructure on the server side, the present invention could not easily be used to enter data by users of systems with such limited technical capabilities on the client, because data entered and selected using the present invention would not be communicated to the existing application/web server. However, the modified invention and method described in **FIG. 9**, referred to as an Application Proxy, offers a solution.

[0208] Although the system depicted in **FIG. 9** can be used to support clients in practically any server-based application server, and particularly in the case of a web server hosting an application used by end users to enter data that is partially retrieved using the present invention, the system is not limited to the web. The system provides an ideal solution for current web-based applications that consist of web browsers **903** on the client side and web host computers **901** with web server software **917** on the server side. To allow the web server **917** to access data selected using the present invention, this system provides a link between the web server and the QuestObjects Server **902**. In this case, QuestObjects Server acts as a data-entry proxy between the existing client system (web browser) and the existing web server. Data entered by the client is submitted to the QuestObjects Adaptor instead of to the web server. The QuestObjects Adaptor then fills in the values of the Questers and passes the data to the web server. An Application Proxy is not required if the QuestObjects Client components can directly insert data into the client entry form on the web browser, as is the case on certain platforms that allow integration between Java applets or other components and JavaScript in the web browser.

[0209] In **FIG. 9**, the web server runs on a host computer **901** typically associated with a fixed IP address or an Internet host name. The web server is accessed by any number of clients using web browsers **903**. To allow users to enter data and send data to the server, web pages make use of HTML forms **904**. To use the present invention, user interface elements such as entry fields in these HTML forms are associated with Questers **905** in the form of browser Plug-ins or Java Applets. Through a QuestObjects Controller **906** those Questers allow the user to access one or more QuestObjects Services hosted by a QuestObjects Server **902** using the protocol of the present invention **907**. The Server Controller **908** forwards user actions generated in the Client Questers **905** to their corresponding Server Questers **909** that thus are always aware of data selected in the Client. When a Server Quester is first activated, it checks whether it is being used by a client system that requires the use of an Application Proxy. If the answer is yes, then the Quester creates a corresponding AppHost Synchronizer **911** that contacts the QuestObjects Adaptor **914** on the host computer **901** using a standardized protocol **915**. The QuestObjects Adaptor then knows which QuestObjects Server to contact to retrieve QuestObjects data **915** after the user submits form data **912** to the application host using the existing application protocol **913**, such as HTTP POST or HTTP GET. The QuestObjects Adaptor then replaces the appropriate form field data with the strings selected in the Server Questers **909** before forwarding this form data, now including data selected using the present invention, to the web server **917**.

DESIGN IMPLEMENTATION

[0210] The preceding detailed description illustrates software objects and methods of a system implementing the present invention. By providing a simple and standardized interface between Client components and any number of Content Engines that accept string-based queries, the present invention gives content publishers, web publishers and software developers an attractive way to offer unprecedented interactive, speedy, up-to-date and controlled access to content without the need to write an access mechanism for each content source.

[0211] In addition to acting as a standardized gateway to any content engine, the present invention can intelligently cache query results, distribute Services over a network of Servers, validate user and other client input, authorize user access and authenticate client software components as needed. These and other optional services are provided by the present invention without requiring additional work on the part of software developers or content publishers. Publishers can also keep track of usage statistics, on a per-user basis as required allowing flexible billing of content access. Content Access Modules allow software developers and vendors of Content Engines such as database vendors and search engine vendors to create simplified ways for developers and implementers of such content engines to disclose information through the present invention.

[0212] End users of the present invention experience an unprecedented level of user-friendliness accessing information that is guaranteed to be up-to-date while being efficiently cached for speedy access as the number of simultaneous users grows.

[0213] The present invention can be implemented on any client and server system using any combination of operating systems and programming languages that support asynchronous network connections and preferably but not necessarily preemptive multitasking and multithreading. The interface of the present invention as it appears to the outside world (i.e. programmers and developers who provide access to end users and programmers who provide Content Access Modules to Content Engines used by content publishers) is independent of both the operating systems and the programming languages used. Adapters can be built allowing the tiers of the system to cooperate even if they use a different operating system or a different programming language. The protocol of the present invention can be implemented on top of networking standards such as TCP/IP. It can also take advantage of inter-object communication standards such as CORBA and DCOM. The object model of the present invention can be mapped to most other programming languages, including Java, C++, Objective C and Pascal.

[0214] Third-party vendors of software development and database management tools can create components that encapsulate the present invention so that users of those tools can access its functionality without any knowledge of the underlying protocols and server-side solutions. For example, a 4GL tool vendor can add an 'auto-complete field' to the toolbox of the development environment allowing developers to simply drop a Questlet into their application. In order to function correctly, the auto-complete field would only need a reference to the QuestObjects Server and one or more QuestObjects Services, but it would not require any additional programming.

[0215] Examples of Applications in which the invention may be used include: Access system for database fields (for lookup and auto-complete services); Enterprise thesauri system; Enterprise search and retrieval systems; Enterprise reference works; Enterprise address books; Control systems for sending sensor readings to a server that responds with appropriate instructions or actions to be taken; Client access to dictionary, thesaurus, encyclopedia and reference works; Access to commercial products database; Literary quotes library; Real-time stock quote provision; Access to real-time news service; Access to Internet advertisements; Access to complex functions (bank check, credit card validation, etc); Access to language translation engines; Access to classification schemes (eg, Library of Congress Subject Headings); Access to lookup lists such as cities or countries in an order form; Personal address books; and, Personal auto-complete histories.

[0216] The foregoing description of preferred embodiments of the present invention has been provided for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Obviously, many modifications and variations will be apparent to the practitioner skilled in the art. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, thereby enabling others skilled in the art to understand the invention for various embodiments and with various modifications that are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalence.

What is claimed is:

1. A system for session-based retrieval at a client system of content from a server system, said server system serving a string-based content, said string-based content including a plurality of strings, comprising:

a communication protocol that provides a session-based connection between a client system and a server system, and allows said client system to query said server system for content;

a client object, in communication with a client software at said client system, said client object capable of transmitting to a server object a plurality of queries to retrieve content from said content engine, wherein each of said plurality of queries comprises a single string character, and wherein each subsequent of said plurality of queries extends the query; and,

a server object, in communication with a server software at said server system, said server object furthermore in communication with said client object via said communication protocol, said server object records each of said plurality of queries, and in response to each of said queries returns increasingly appropriate content information to the client object as the query is being extended.

2. The system of claim 1 wherein said client software operates on or at a first computer and said server software operates on or at a second computer, and wherein both of said first and said second computers are connected via a communication network protocol.

3. The system of claim 1 wherein said server software and said client software runs on the same computer.

4. The system of claim 1 wherein said server software runs on a plurality of separate computers, and wherein said client queries are distributed over said separate servers.

5. The system of claim 1 wherein said server software stores previously used strings and returns said stored strings to the client in response to new client queries, without accessing said content engine.

6. The system of claim 1 wherein said client software is embedded into a software application that provides a visual interface to an operator.

7. The system of claim 1 wherein said client software is used as a content engine for another software system.

8. The system of claim 1 wherein said client software accumulates a plurality of said single character queries as they are entered into the client, before sending them together to said server software as a string.

9. The system of claim 1 wherein said client software stores previously received responses and uses these as the response to a new query by the user, without re-accessing the server.

10. The system of claim 1 wherein said client software stores a pre-defined string and automatically transmits it to the server as the client software is first accessed, and wherein additional entry of query characters is not required before server responses are sent to the client.

11. The system of claim 1 wherein said server software stores the state of query and response of the client software, and restores the state of the client software after any interruption in said communication protocol.

12. The system of claim 1 where said client software adds a qualifier to the query that is passed to the content engine by the server, whereby the content engine can use said qualifier to execute the query and return appropriate results based on both the query string and its qualifier.

13. The system of claim 1 where said client software identifies a user of the system to the server software whereby the server can store statistics and provides a history of queries and corresponding responses appropriate to said user.

14. The system of claim 1 where said server software is distributed within a server tier and a syndication tier, and wherein said client software communicates to the server tier on a single computer, and wherein each query is forwarded by the server tier to an appropriate syndicate of content channels connected to the server tier on a different computer.

15. The system of claim 1 where said server software applies a content engine dependent pattern and filter to characters received from the client before queries are transmitted to the content engine.

16. The system of claim 15 wherein the number of queries transmitted to the content engine is limited.

17. The system of claim 1 where server responses comprise lists of strings, wherein each string is accompanied by corresponding metadata, whereby the metadata contains logical links to other data sources or Uniform Resource Identifiers.

18. The system of claim 16 where each string in the server response list is a link to another data source or a Uniform Resource Identifier.

19. A system for retrieval of content in a distributed client-server system, comprising:

a content engine, for providing string-based content;

a session protocol, for providing a session-based connection between a client system and a server system;

a client object, at said client system, for transmitting to said server system a successive plurality of queries to retrieve content from said content engine, wherein a first of said plurality of queries initially comprises a single character, and wherein each successive query thereafter extends the query; and,

a server object, at said server system, in communication with said client object via said session protocol, for recording each of said plurality of single character queries, and in response to each successive query returns increasingly content to the client object appropriate to the extended query.

20. A system for session-based retrieval of content from a string-based content engine system, comprising:

a user interface, for inputting a plurality of queries to a client object, for subsequent transmission of said plurality of queries to a remote server object, wherein said remote server object is in communication with said content engine, wherein further each of said plurality of queries comprises a single string character;

a session protocol manager that maintains a session between said client object and said server object;

a client object, in communication with said user interface, for transmitting to said remote server object, during a session, a subset of said plurality of queries, and for receiving from said server object, content information appropriate to said session and to said subset of queries; and,

an input status mechanism for visually indicating the status of said content information appropriate to said session.

21. A system for session-based delivery of content from a string-based content engine to a client, comprising:

a server, for receiving a request for content from a client object at said client, said request comprising a plurality of single string character queries;

a session protocol manager that maintains a session between said client object and said server object; and,

a server object in communication with said server, for providing content information appropriate to said session, said server object records each of said plurality of single character queries, and in response to each of said queries returns increasingly appropriate content information to the client as the query is being extended.

22. A system for session-based retrieval at a client system of a string-based content from a server system, comprising:

a content engine, for serving a string-based content, said string-based content including a plurality of strings;

a communication protocol that provides a session-based connection between a client system and a server system, and allows said client system to query said content engine;

a client object, in communication with a client software application, said client object capable of transmitting to a server object a plurality of queries to retrieve content from said content engine, wherein each of said plurality of queries comprises a single string character, and wherein each subsequent of said plurality of queries extends the query; and,

a server object in communication with said client object via a communication protocol, said server object records each of said plurality of single character queries, and in response to each of said queries returns increasingly appropriate content information to the client object as the query is being extended.

23. A system for session-based retrieval at a client system of a string-based content from a server system, comprising:

a content engine, for serving a string-based content, said string-based content including a plurality of strings;

a communication protocol that provides a session-based connection between a client system and a server system, and allows said client system to query said content engine;

a client object, in asynchronous communication with a client software application, said client object capable of continuously transmitting to a server object a plurality of queries to retrieve content from said content engine, wherein each of said plurality of queries comprises a single string character, and wherein each subsequent of said plurality of queries extends the query; and,

a server object, in asynchronous communication with said client object via said communication protocol, said server object continuously records each of said plurality of single character queries, and in response to each of said queries returns increasingly appropriate content information to the client object as the query is being extended.

24. A user interface mechanism, for use with a client application of a session-based content retrieval system, said user interface mechanism indicating both the availability of a session between said client application and a remote content server, and the status of said session, said mechanism comprising:

a user interface element, in communication with said client application, said user interface element allows a user to input data for transmission to a remote content server;

a session indicator, said session indicator displayed within a first portion of said user interface element, for indicating the presence of a session between said client application and said content server; and,

a status indicator, said status indicator displayed within a second portion of said user interface element, for indicating the status of available content at said content server for selection by said user at said user interface element.

25. The mechanism of claim 24, wherein said user interface element is an application input field.

26. The mechanism of claim 24, wherein said session indicator displays a triangular display element to indicate the presence of said session, and does not display said triangular display element to indicate the absence of said session.

27. The mechanism of claim 24, wherein said status indicator displays one, or a plurality of, arrow display elements to indicate the transfer of data from said client application to said server during said session, and the presence of available session-specific content at said server.

* * * * *

B

US 20060075120A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2006/0075120 A1**

Smit (43) **Pub. Date: Apr. 6, 2006**

(54) **SYSTEM AND METHOD FOR UTILIZING ASYNCHRONOUS CLIENT SERVER COMMUNICATION OBJECTS**

(76) Inventor: **Mark H. Smit**, Maarssen (NL)

Correspondence Address:
FLIESLER MEYER, LLP
FOUR EMBARCADERO CENTER
SUITE 400
SAN FRANCISCO, CA 94111 (US)

(21) Appl. No.: **11/257,912**

(22) Filed: **Oct. 25, 2005**

Related U.S. Application Data

(63) Continuation-in-part of application No. 09/933,493, filed on Aug. 20, 2001.

(60) Provisional application No. 60/622,907, filed on Oct. 28, 2004.

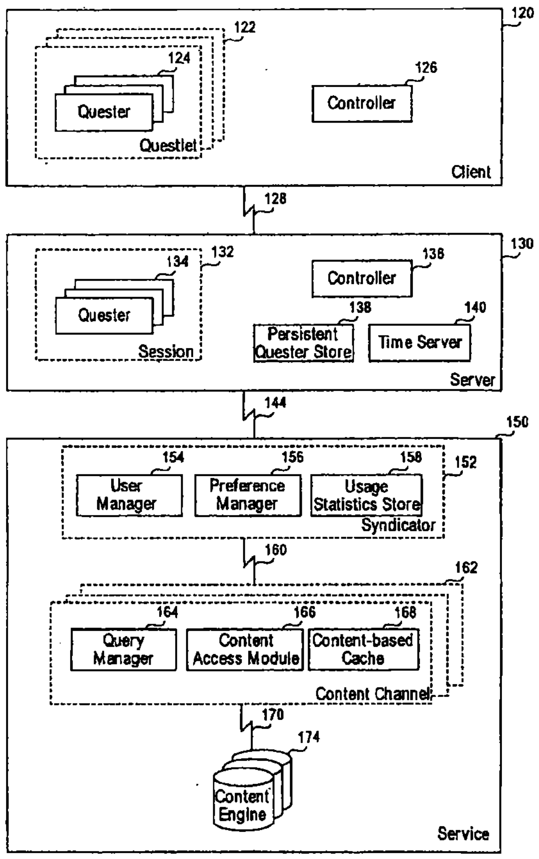
Publication Classification

(51) **Int. Cl.**
G06F 15/16 (2006.01)

(52) **U.S. Cl.** **709/227**

(57) **ABSTRACT**

A session-based client-server asynchronous information search and retrieval system for sending character-by-character or multi-character strings of data to an intelligent server, that can be configured to immediately analyze the lengthening string and return to the client increasingly appropriate search information. Embodiments include integration within an Internet, web or other online environment, including applications for use in interactive database searching, data entry, online searching, online purchasing, music purchasing, people-searching, and other applications. In some implementations the system may be used to provide dynamically focused suggestions, auto-completed text, or other input-related assistance, to the user.



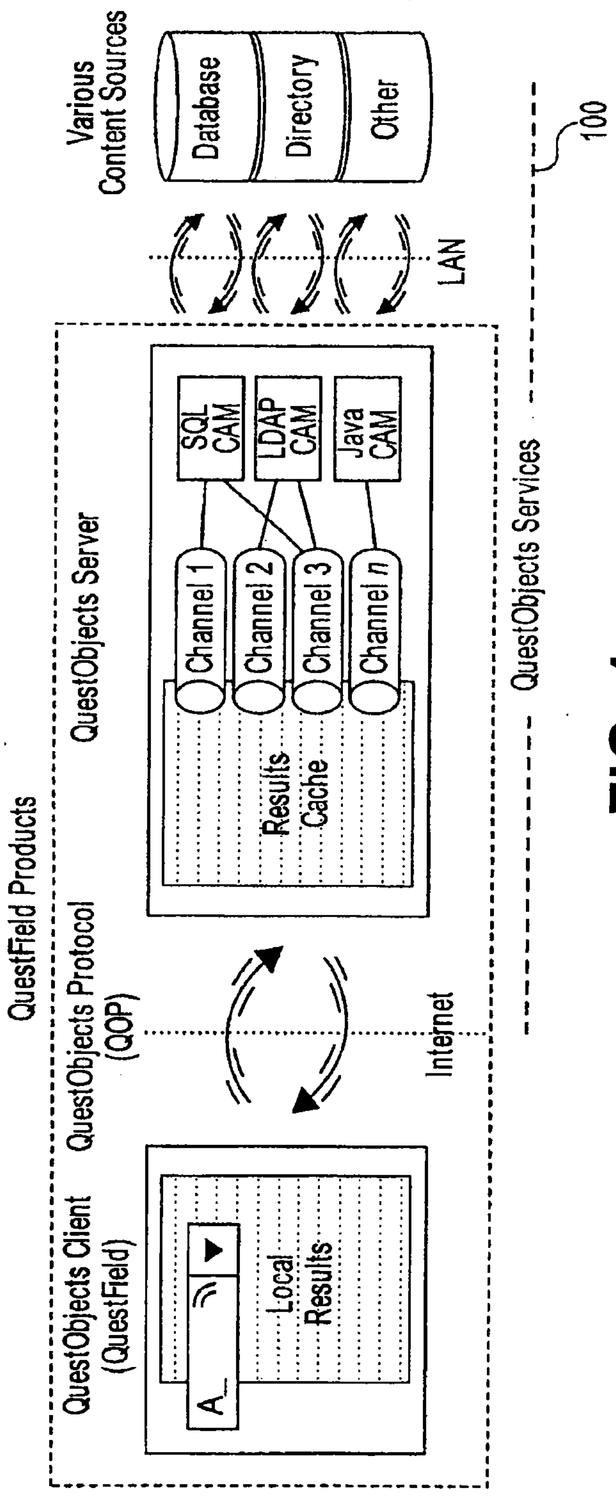

QuestField Products
QuestObjects Client (QuestField)
A_
Local Results
QuestObjects Protocol (QOP)
Internet
QuestObjects Server
Results Cache
Channel 1
Channel 2
Channel 3
Channel n
SQL CAM
LDAP CAM
Java CAM
LAN
Various Content Sources
Database
Directory
Other
QuestObjects Services
100

FIG. 1

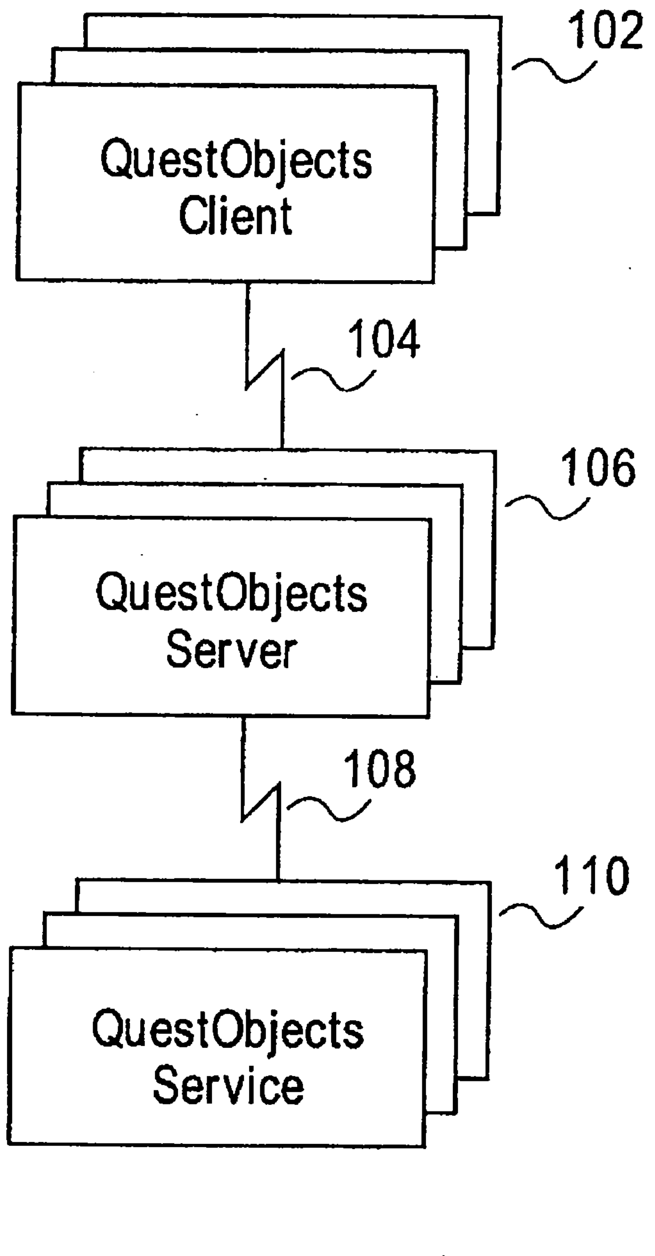

102
QuestObjects Client
104
106
QuestObjects Server
108
110
QuestObjects Service

FIG. 2

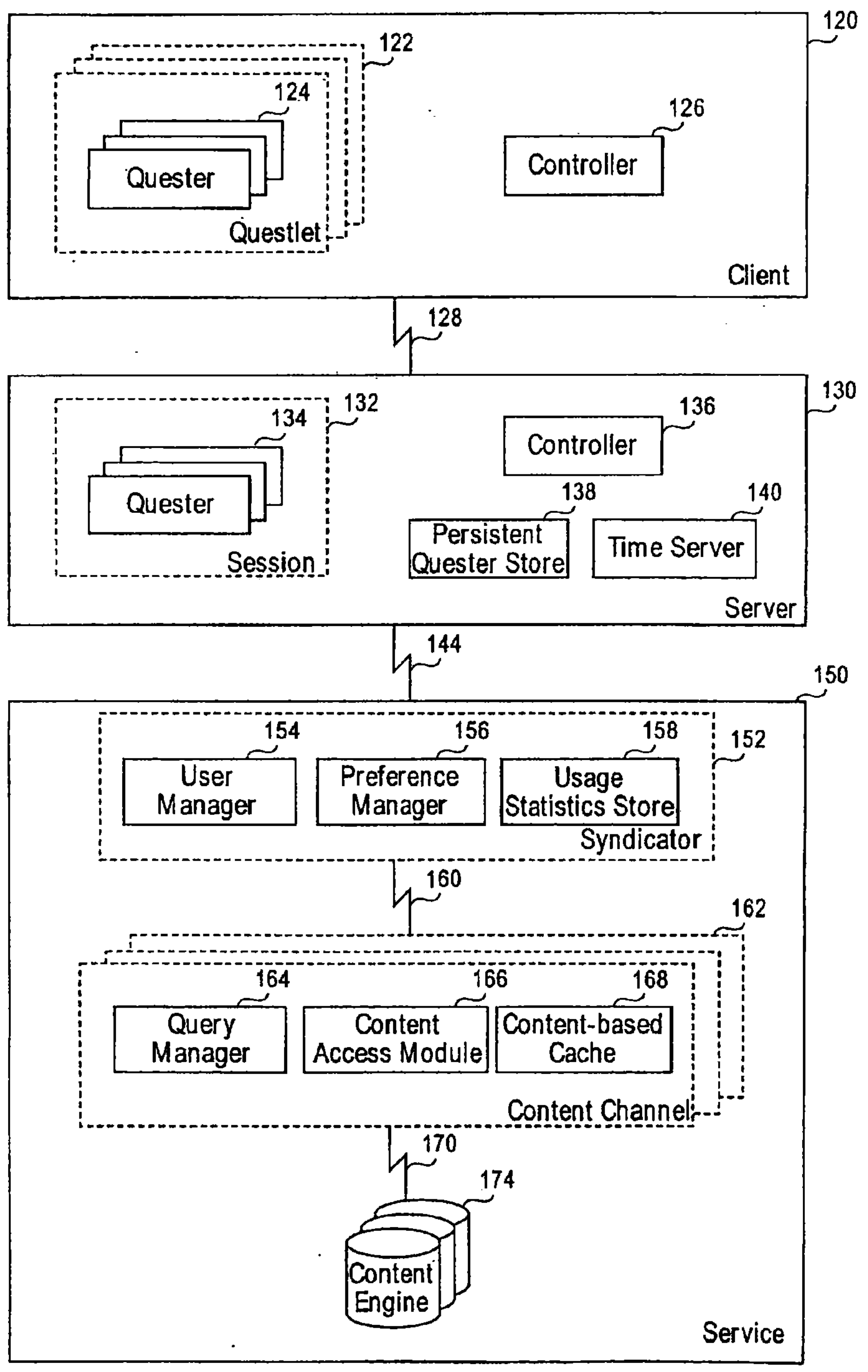

120
122
124
Quester
Questlet
126
Controller
Client
128
130
132
134
Quester
Session
136
Controller
138
Persistent Quester Store
140
Time Server
Server
144
150
154
User Manager
156
Preference Manager
158
Usage Statistics Store
152
Syndicator
160
162
164
Query Manager
166
Content Access Module
168
Content-based Cache
Content Channel
170
174
Content Engine
Service

FIG. 3

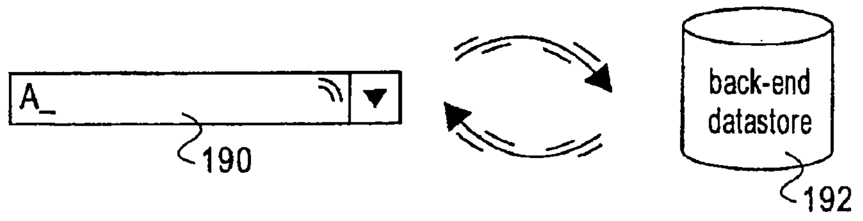


FIG. 4

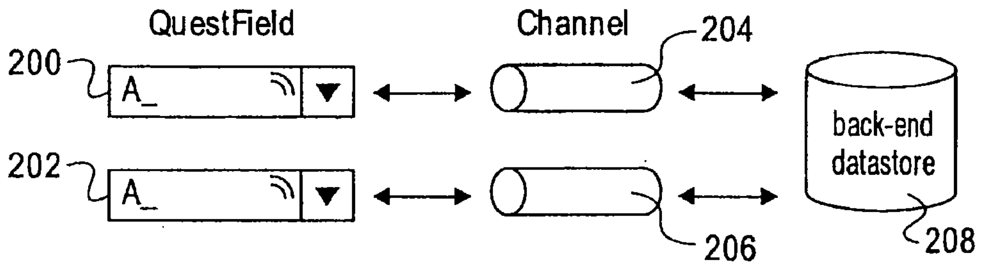


FIG. 5

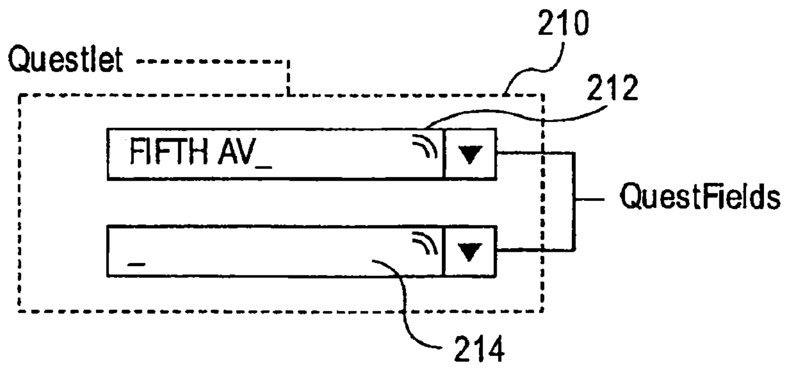


FIG. 6

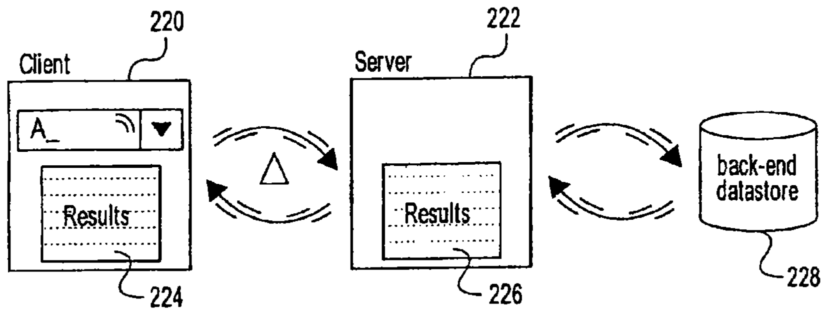

220
Client
A_
Results
224
222
Server
Results
226
back-end datastore
228

FIG. 7

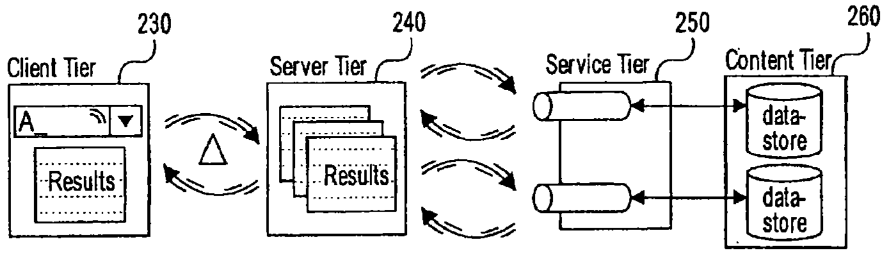

230
Client Tier
A_
Results
240
Server Tier
Results
250
Service Tier
260
Content Tier
data-store
data-store

FIG. 8

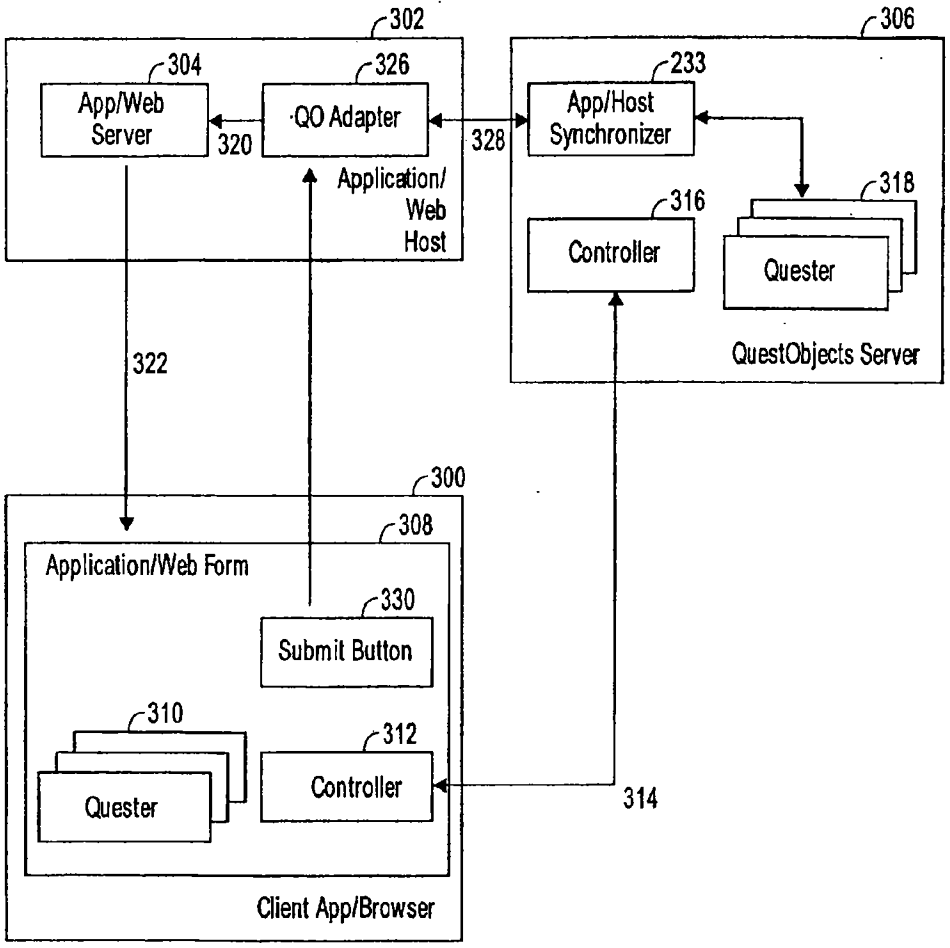

302
304
App/Web Server
326
QO Adapter
320
Application/ Web Host
306
233
App/Host Synchronizer
328
316
Controller
318
Quester
QuestObjects Server
322
300
308
Application/Web Form
330
Submit Button
310
Quester
312
Controller
314
Client App/Browser

FIG. 9

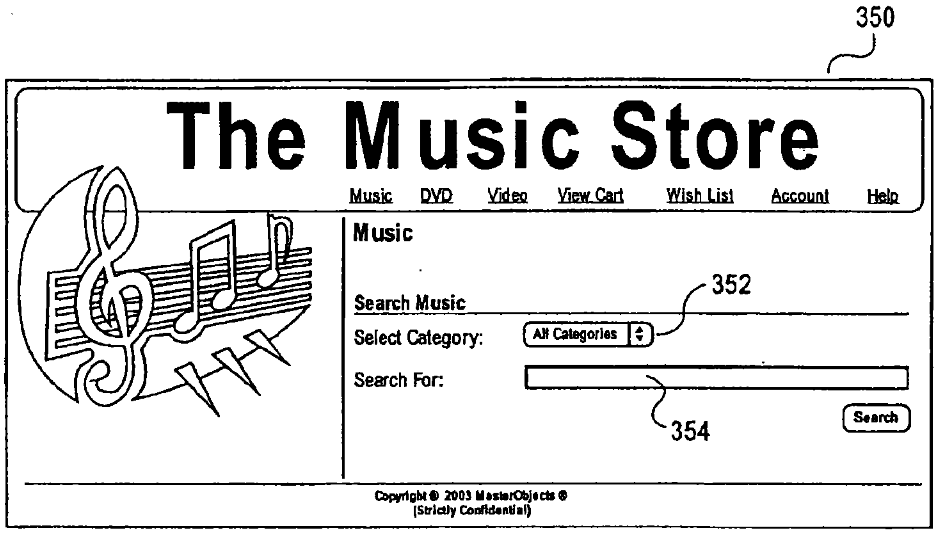


FIG. 10

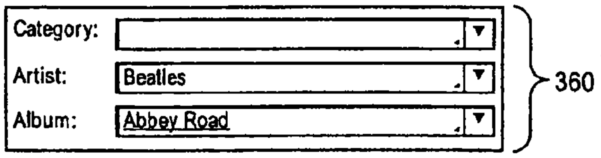

Category:
Artist: Beatles
Album: Abbey Road

FIG. 11

```
<script type="text/javascript" language="javascript"
            src="qo-common.js"></script>

<script type="text/javascript" language="javascript"
            src="scripts/qo-questlet.js"></script>

<script type="text/javascript" language="javascript">

<!-
    var config = {};
    config.connectString ="/qo-server";
    config.contentChannel  ="artist-name";
    config.postString   ="/your-application";
    config.helpLink  = "http://www.questobjects.com/";
    config.helpTarget  ="help";
    config.helpParameters  ="top=30,left=30,width=760,height=560";
    config.questFieldName  ="ArtistField";
    config.questFieldFile  ="qo-autocomplete-questfield.swf";
    config.questFieldWidth =250;
    config.questFieldHeight =20;
    config.popUpFile  ="qo-autocomplete-popup.swf";
    config.popUpWidth  =350;
    config.popUpHeight  =(14*17)+4;
    config.popUpToLeft  =false;
    config.bgColor   ="#FFFFFF";
    qoInsertPopUpQuestField (config);

//->
</script>
```

370

FIG. 12

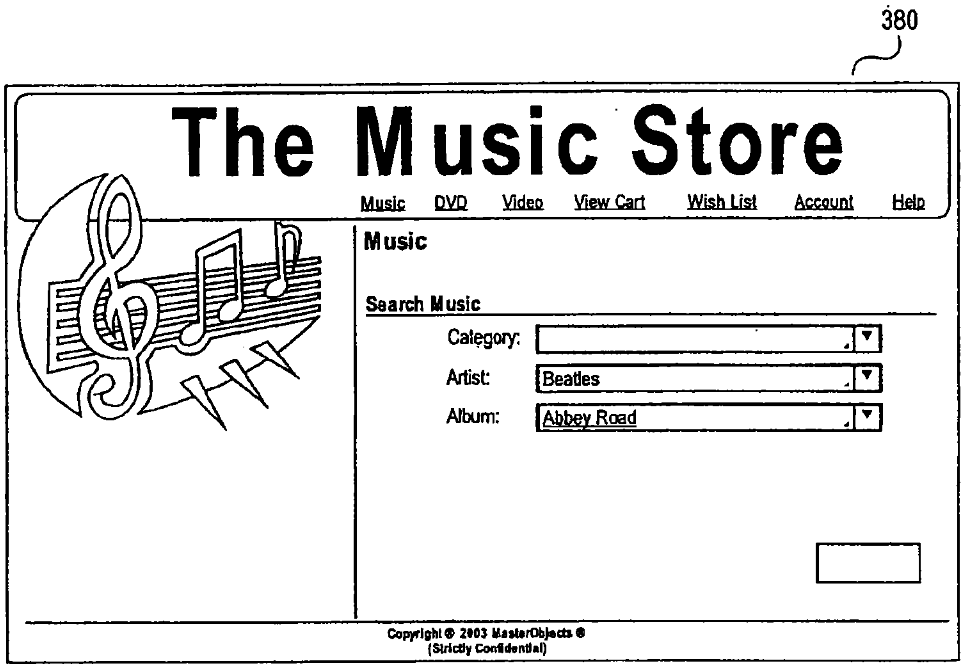
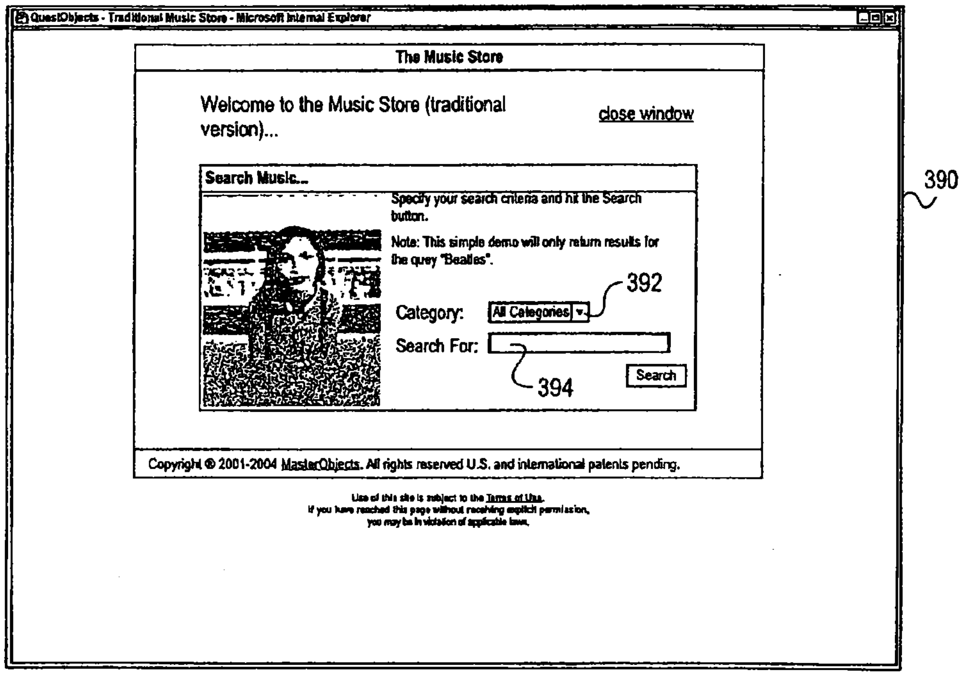

380
The Music Store
Music DVD Video View Cart Wish List Account Help
Music
Search Music
Category:
Artist:
Beatles
Album:
Abbey Road
Copyright © 2003 MasterObjects ®
(Strictly Confidential)

FIG. 13



QuestObjects - Traditional Music Store - Microsoft Internet Explorer

The Music Store

Welcome to the Music Store (traditional version)...

close window

Search Music...

Specify your search criteria and hit the Search button.

Note: This simple demo will only return results for the query "Beatles".

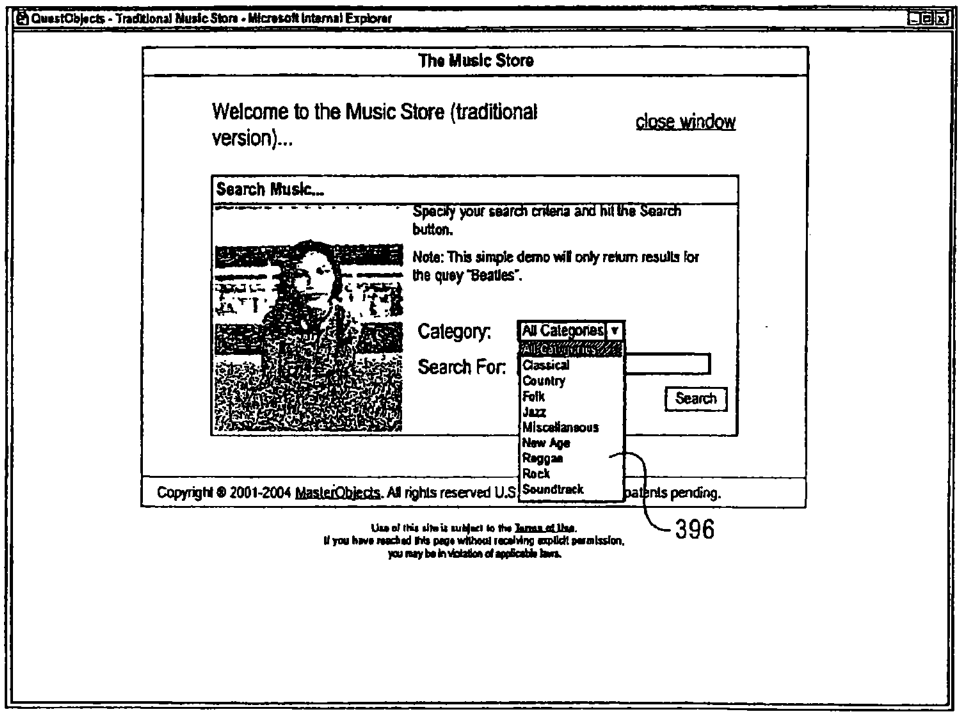
Category: All Categories

Search For:

Copyright © 2001-2004 MasterObjects. All rights reserved U.S. and international patents pending.

Use of this site is subject to the Terms of Use.
If you have reached this page without receiving explicit permission,
you may be in violation of applicable laws.

FIG. 14


QuestObjects - Traditional Music Store - Microsoft Internet Explorer

The Music Store

Welcome to the Music Store (traditional version)...

close window

Search Music...

Specify your search criteria and hit the Search button.

Note: This simple demo will only return results for the quey "Beatles".

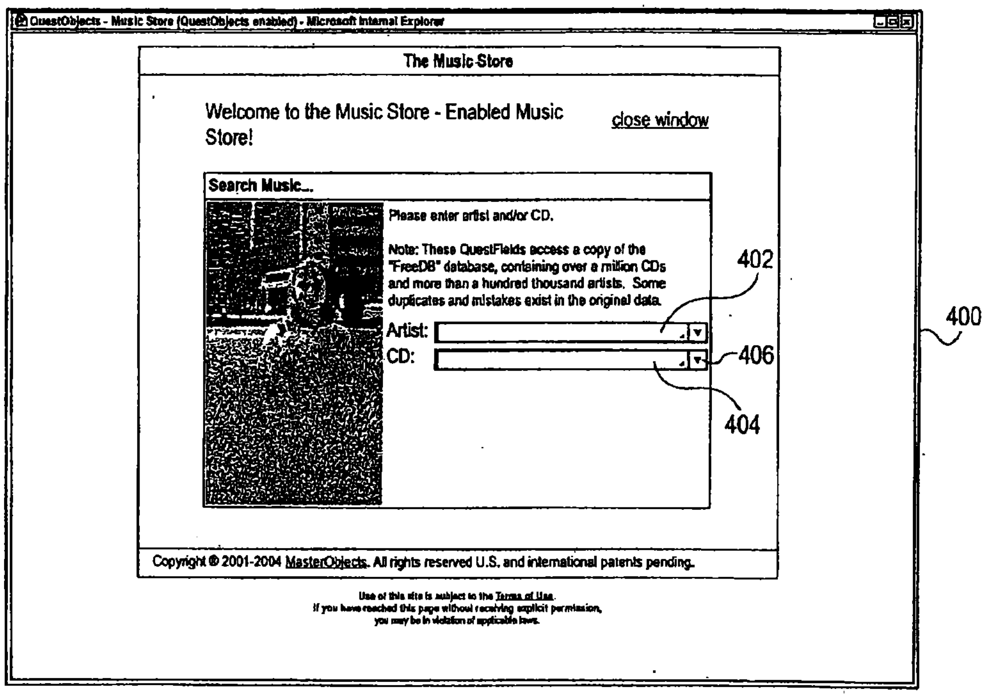
Category: All Categories

Search For:

Copyright © 2001-2004 MasterObjects. All rights reserved U.S. patents pending.

Use of this site is subject to the Terms of Use.
If you have reached this page without receiving explicit permission, you may be in violation of applicable laws.

FIG. 15


QuestObjects - Music Store (QuestObjects enabled) - Microsoft Internal Explorer
The Music Store
close window
Search Music...
Artist:
CD:
402
400
406
404

Welcome to the Music Store - Enabled Music Store!

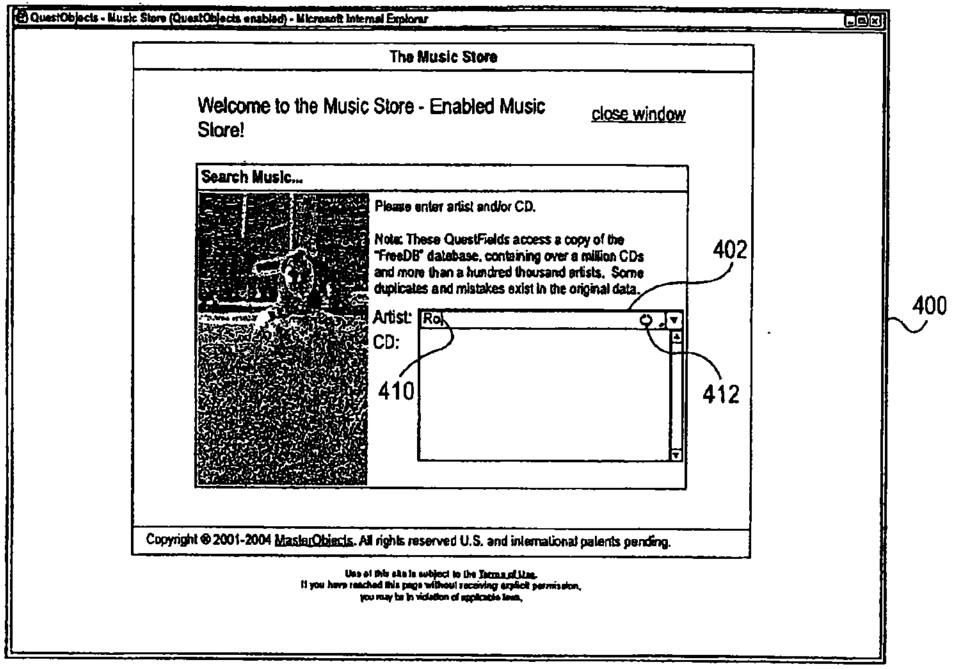
Please enter artist and/or CD.

Note: These QuestFields access a copy of the "FreeDB" database, containing over a million CDs and more than a hundred thousand artists. Some duplicates and mistakes exist in the original data.

Copyright © 2001-2004 MasterObjects. All rights reserved U.S. and international patents pending.

Use of this site is subject to the Terms of Use.
If you have reached this page without receiving explicit permission, you may be in violation of applicable laws.

FIG. 16



QuestObjects - Music Store (QuestObjects enabled) - Microsoft Internet Explorer

The Music Store

Welcome to the Music Store - Enabled Music Store!

close window

Search Music...

Please enter artist and/or CD.

Note: These QuestFields access a copy of the "FreeDB" database, containing over a million CDs and more than a hundred thousand artists. Some duplicates and mistakes exist in the original data.

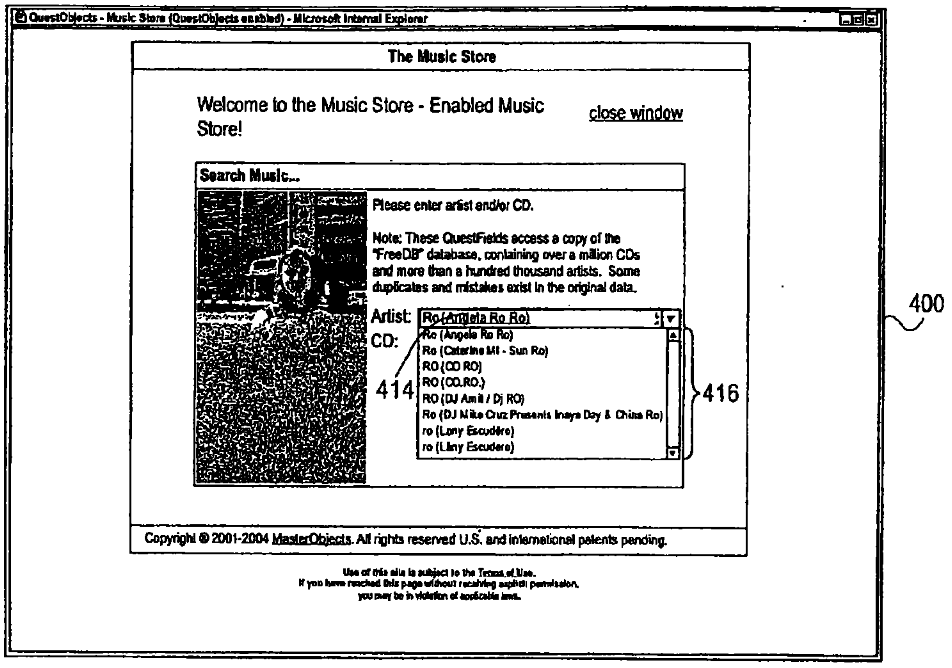
Artist: Ro

CD:

Copyright © 2001-2004 MasterObjects. All rights reserved U.S. and international patents pending.

Use of this site is subject to the Terms of Use.
If you have reached this page without receiving explicit permission,
you may be in violation of applicable laws.

FIG. 17


QuestObjects - Music Store (QuestObjects enabled) - Microsoft Internet Explorer
close window
Search Music...
400
414
416

The Music Store

Welcome to the Music Store - Enabled Music Store!

Please enter artist and/or CD.

Note: These QuestFields access a copy of the "FreeDB" database, containing over a million CDs and more than a hundred thousand artists. Some duplicates and mistakes exist in the original data.

Artist: Ro (Angela Ro Ro)

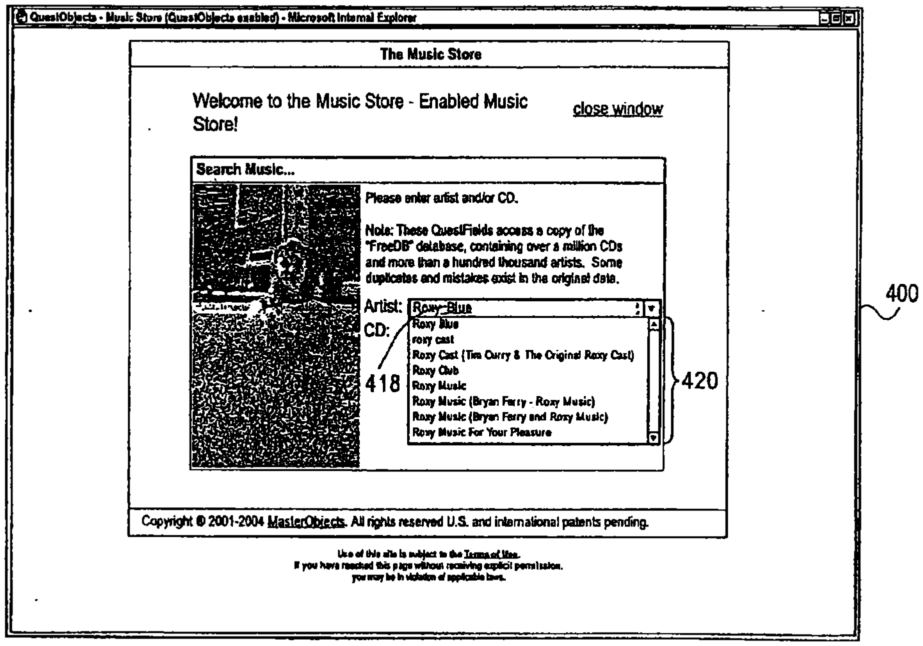
CD:

Ro (Angela Ro Ro)
Ro (Caterina MI - Sun Ro)
RO (CD RO)
RO (CD.RO.)
RO (DJ Amit / Dj RO)
Ro (DJ Mike Cruz Presents Inaya Day & China Ro)
ro (Lony Escudéro)
ro (Liny Escudero)

Copyright © 2001-2004 MasterObjects. All rights reserved U.S. and international patents pending.

Use of this site is subject to the Terms of Use.
If you have reached this page without receiving explicit permission,
you may be in violation of applicable laws.

FIG. 18



QuestObjects - Music Store (QuestObjects enabled) - Microsoft Internal Explorer

The Music Store

Welcome to the Music Store - Enabled Music Store!

close window

Search Music...

Please enter artist and/or CD.

Note: These QuestFields access a copy of the "FreeDB" database, containing over a million CDs and more than a hundred thousand artists. Some duplicates and mistakes exist in the original data.

Artist: Roxy Blue

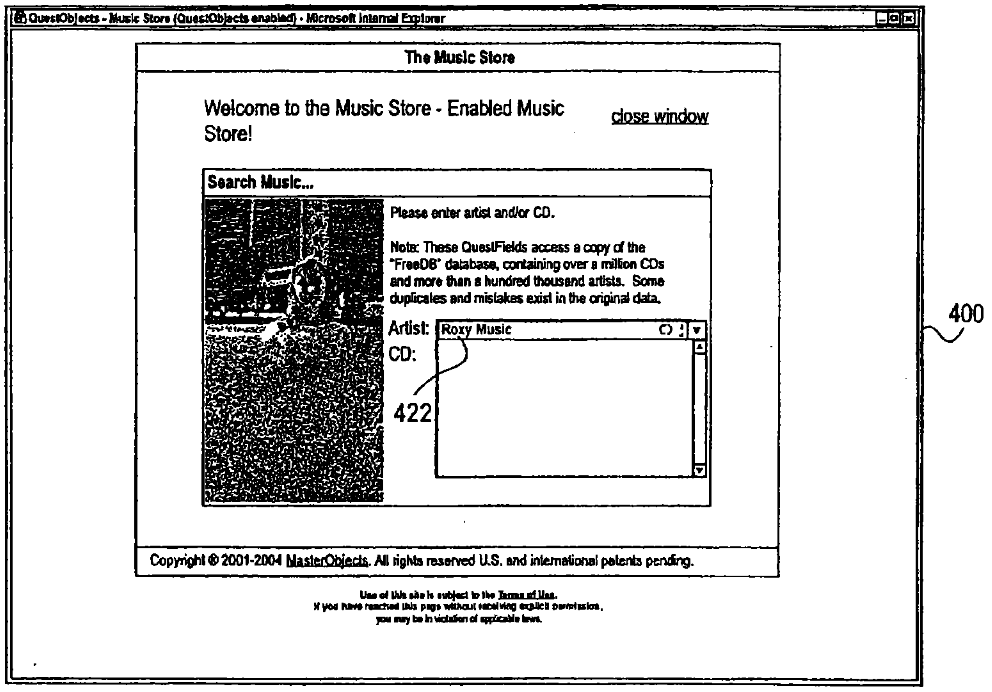
CD:

Roxy Blue
roxy cast
Roxy Cast (Tim Curry & The Original Roxy Cast)
Roxy Club
Roxy Music
Roxy Music (Bryan Ferry - Roxy Music)
Roxy Music (Bryan Ferry and Roxy Music)
Roxy Music For Your Pleasure

Copyright © 2001-2004 MasterObjects. All rights reserved U.S. and international patents pending.

Use of this site is subject to the Terms of Use.
If you have reached this page without receiving explicit permission, you may be in violation of applicable laws.

FIG. 19


QuestObjects - Music Store (QuestObjects enabled) - Microsoft Internet Explorer

The Music Store

Welcome to the Music Store - Enabled Music Store!

close window

Search Music...

Please enter artist and/or CD.

Note: These QuestFields access a copy of the "FreeDB" database, containing over a million CDs and more than a hundred thousand artists. Some duplicates and mistakes exist in the original data.

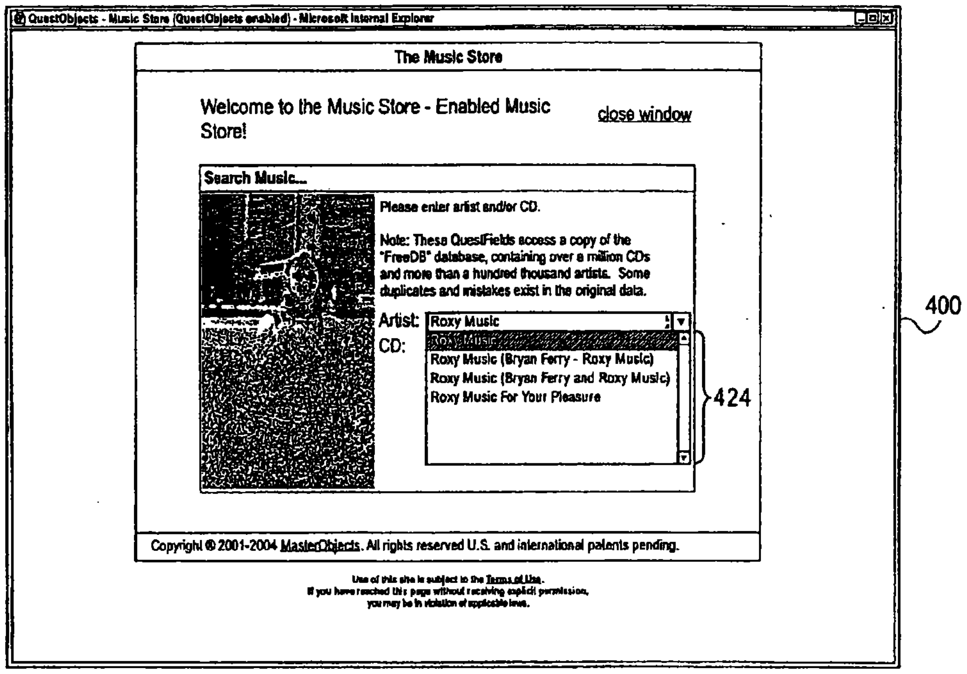
Artist: Roxy Music

CD:

Copyright © 2001-2004 MasterObjects. All rights reserved U.S. and international patents pending.

Use of this site is subject to the Terms of Use.
If you have reached this page without receiving explicit permission,
you may be in violation of applicable laws.

FIG. 20



QuestObjects - Music Store (QuestObjects enabled) - Microsoft Internet Explorer

The Music Store

Welcome to the Music Store - Enabled Music Store!

close window

Search Music...

Please enter artist and/or CD.

Note: These QuestFields access a copy of the "FreeDB" database, containing over a million CDs and more than a hundred thousand artists. Some duplicates and mistakes exist in the original data.

Artist: Roxy Music

CD:

Roxy Music

Roxy Music (Bryan Ferry - Roxy Music)

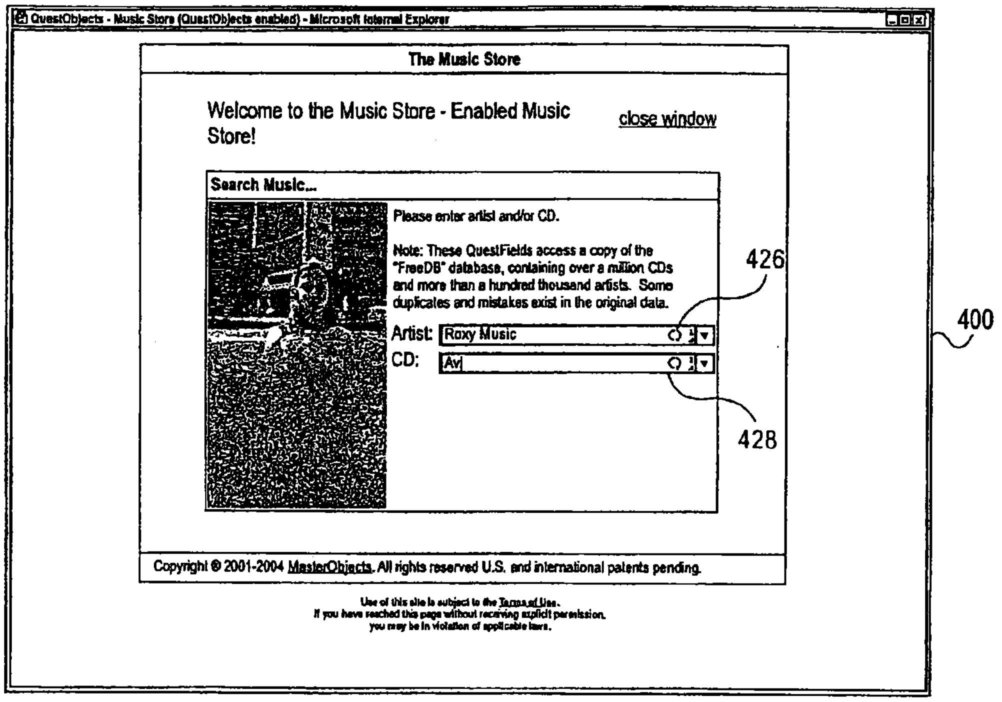
Roxy Music (Bryan Ferry and Roxy Music)

Roxy Music For Your Pleasure

Copyright © 2001-2004 MasterObjects. All rights reserved U.S. and international patents pending.

Use of this site is subject to the Terms of Use.
If you have reached this page without receiving explicit permission,
you may be in violation of applicable laws.

FIG. 21


QuestObjects - Music Store (QuestObjects enabled) - Microsoft Internet Explorer

The Music Store

Welcome to the Music Store - Enabled Music Store!

close window

Search Music...

Please enter artist and/or CD.

Note: These QuestFields access a copy of the "FreeDB" database, containing over a million CDs and more than a hundred thousand artists. Some duplicates and mistakes exist in the original data.

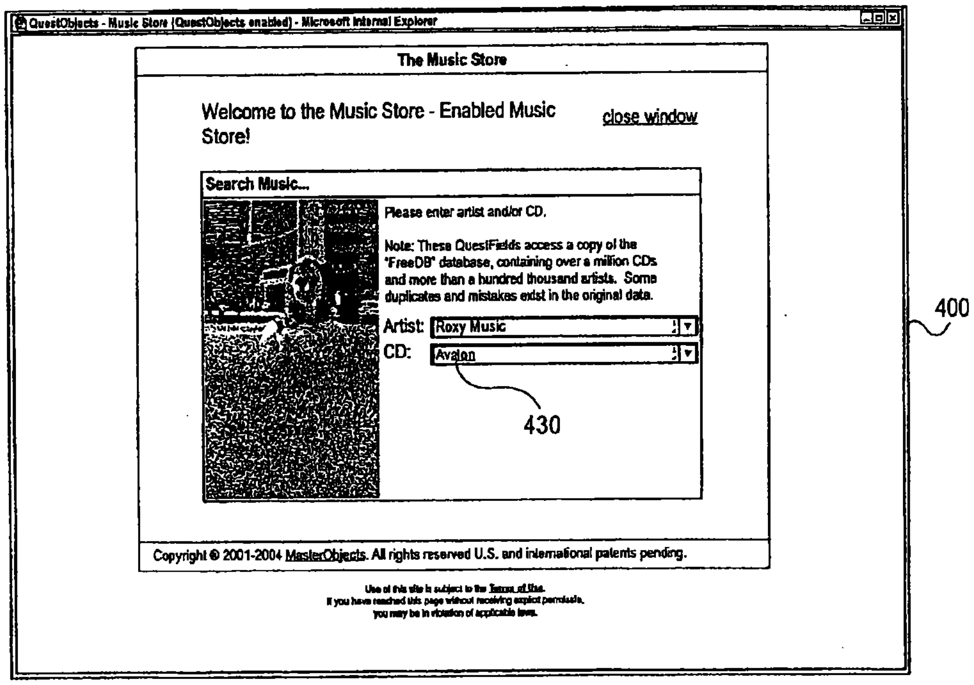
Artist: Roxy Music

CD: Av

Copyright © 2001-2004 MasterObjects. All rights reserved U.S. and international patents pending.

Use of this site is subject to the Terms of Use.
If you have reached this page without receiving explicit permission, you may be in violation of applicable laws.

FIG. 22


QuestObjects - Music Store (QuestObjects enabled) - Microsoft Internet Explorer

The Music Store

Welcome to the Music Store - Enabled Music Store!

close window

Search Music...

Please enter artist and/or CD.

Note: These QuestFields access a copy of the "FreeDB" database, containing over a million CDs and more than a hundred thousand artists. Some duplicates and mistakes exist in the original data.

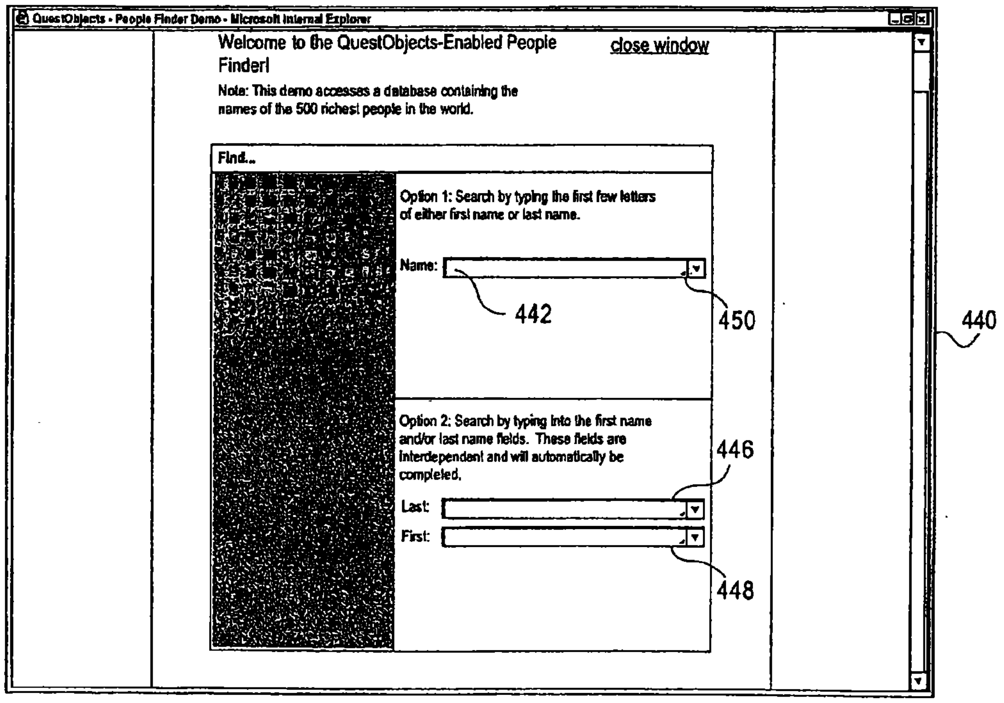
Artist: Roxy Music

CD: Avalon

Copyright © 2001-2004 MasterObjects. All rights reserved U.S. and international patents pending.

Use of this site is subject to the Terms of Use.
If you have reached this page without receiving explicit permission,
you may be in violation of applicable laws.

FIG. 23


QuestObjects - People Finder Demo - Microsoft Internet Explorer
close window
Find...
Name:
Last:
First:
442
450
440
446
448

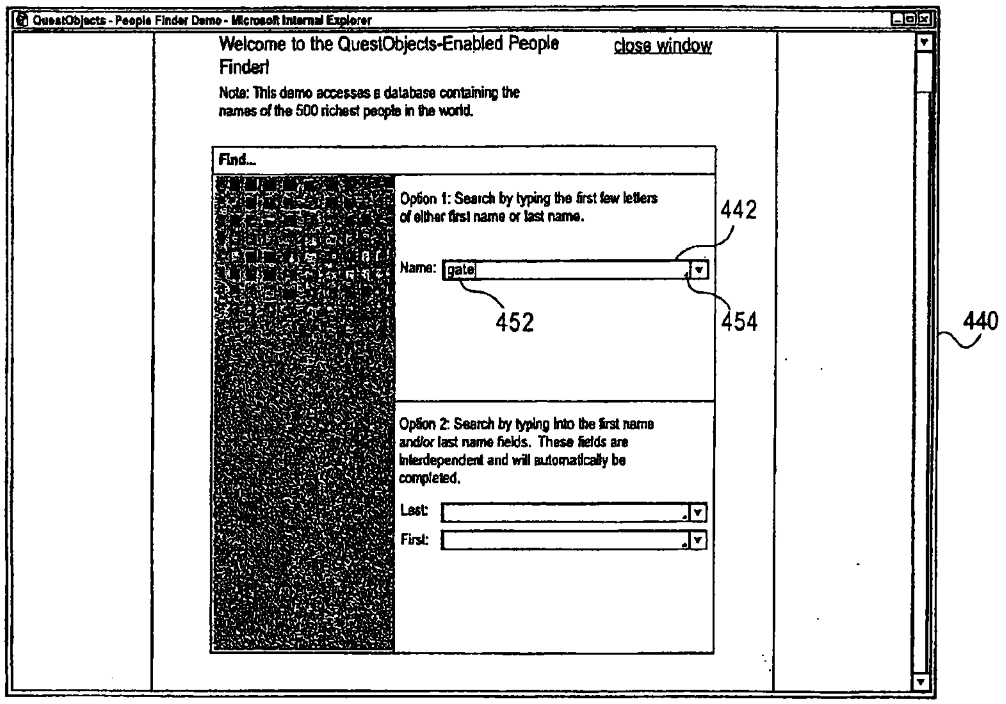
Welcome to the QuestObjects-Enabled People Finder!

Note: This demo accesses a database containing the names of the 500 richest people in the world.

Option 1: Search by typing the first few letters of either first name or last name.

Option 2: Search by typing into the first name and/or last name fields. These fields are interdependent and will automatically be completed.

FIG. 24


QuestObjects - People Finder Demo - Microsoft Internet Explorer

Welcome to the QuestObjects-Enabled People Finder!

close window

Note: This demo accesses a database containing the names of the 500 richest people in the world.

Find...

Option 1: Search by typing the first few letters of either first name or last name.

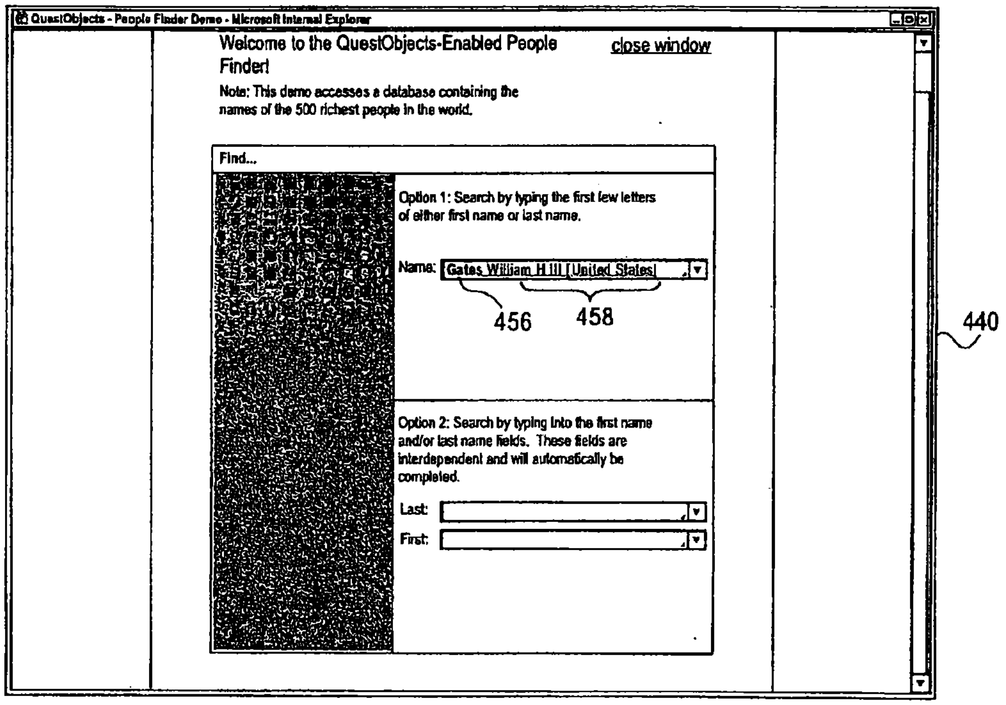
Name: gate

Option 2: Search by typing into the first name and/or last name fields. These fields are interdependent and will automatically be completed.

Last:

First:

FIG. 25


QuestObjects - People Finder Demo - Microsoft Internet Explorer
close window
Find...
Name:
Last:
First:
456
458
440

Welcome to the QuestObjects-Enabled People Finder!

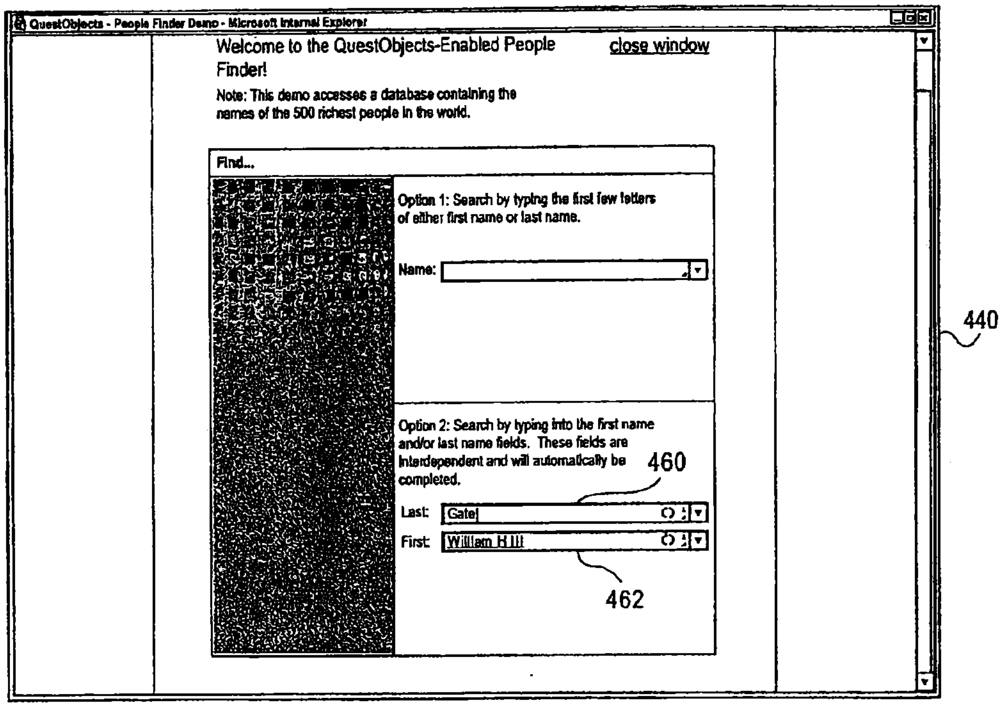
Note: This demo accesses a database containing the names of the 500 richest people in the world.

Option 1: Search by typing the first few letters of either first name or last name.

Name: Gates William H III [United States]

Option 2: Search by typing into the first name and/or last name fields. These fields are interdependent and will automatically be completed.

FIG. 26


QuestObjects - People Finder Demo - Microsoft Internet Explorer
close window
Find...
440
460
462

Welcome to the QuestObjects-Enabled People Finder!

Note: This demo accesses a database containing the names of the 500 richest people in the world.

Option 1: Search by typing the first few letters of either first name or last name.

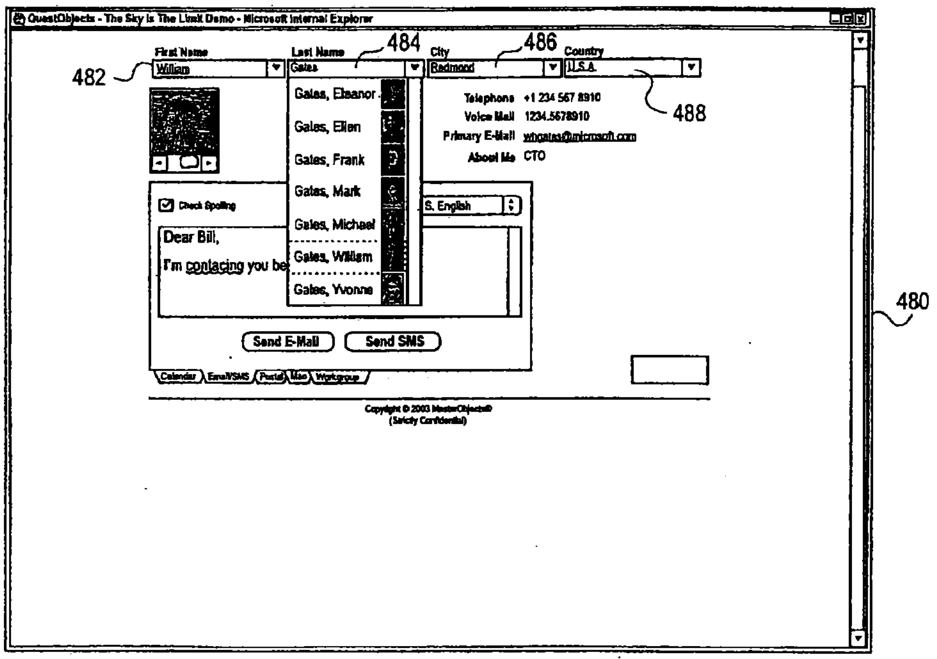
Name:

Option 2: Search by typing into the first name and/or last name fields. These fields are interdependent and will automatically be completed.

Last: Gate

First: William H III

FIG. 27



First Name: William
Last Name: Gates
City: Redmond
Country: U.S.A.

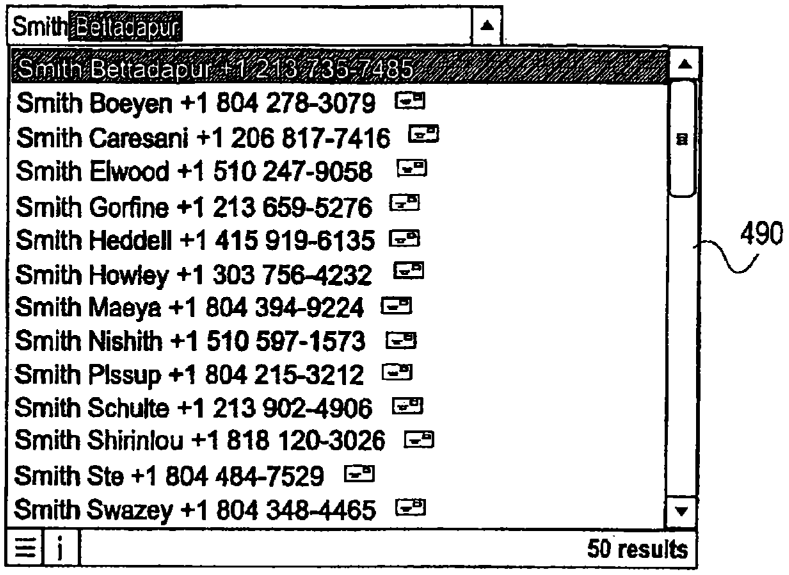
Telephone +1 234 567 8910
Voice Mail 1234.5678910
Primary E-Mail whgates@microsoft.com
About Me CTO

Dear Bill,

I'm contacing you be

Copyright © 2003 MasterObjects®
(Strictly Confidential)

FIG. 28

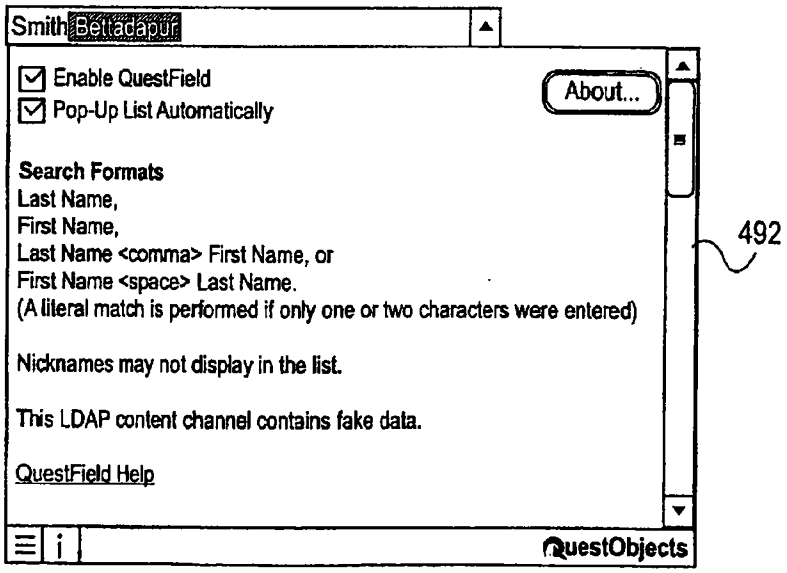


Smith Bettadapur

Smith Bettadapur +1 213 735-7485
Smith Boeyen +1 804 278-3079
Smith Caresani +1 206 817-7416
Smith Elwood +1 510 247-9058
Smith Gorfine +1 213 659-5276
Smith Heddell +1 415 919-6135
Smith Howley +1 303 756-4232
Smith Maeya +1 804 394-9224
Smith Nishith +1 510 597-1573
Smith Plssup +1 804 215-3212
Smith Schulte +1 213 902-4906
Smith Shirinlou +1 818 120-3026
Smith Ste +1 804 484-7529
Smith Swazey +1 804 348-4465

50 results

FIG. 29



Smith Bettadapur

- [✓] Enable QuestField
- [✓] Pop-Up List Automatically

Search Formats
Last Name,
First Name,
Last Name <comma> First Name, or
First Name <space> Last Name.
(A literal match is performed if only one or two characters were entered)

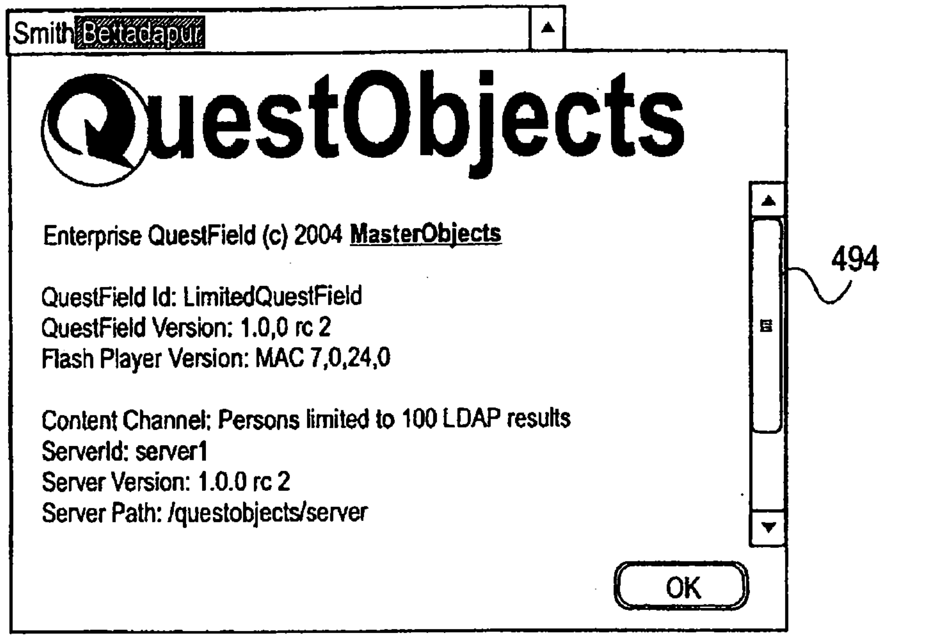
Nicknames may not display in the list.

This LDAP content channel contains fake data.

QuestField Help

QuestObjects

FIG. 30



Enterprise QuestField (c) 2004 MasterObjects

QuestField Id: LimitedQuestField
QuestField Version: 1.0,0 rc 2
Flash Player Version: MAC 7,0,24,0

Content Channel: Persons limited to 100 LDAP results
ServerId: server1
Server Version: 1.0.0 rc 2
Server Path: /questobjects/server

FIG. 31

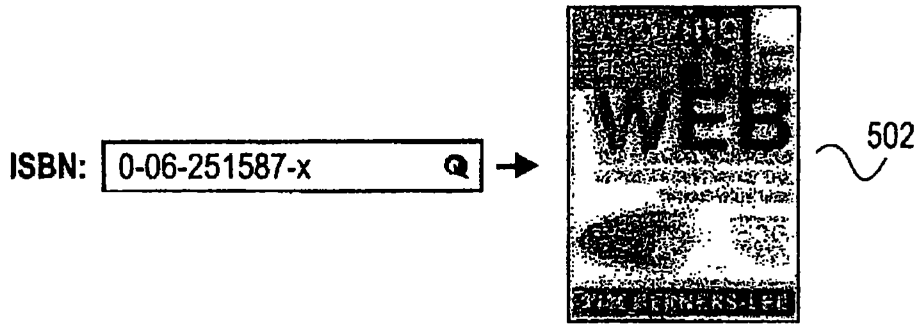


FIG. 32

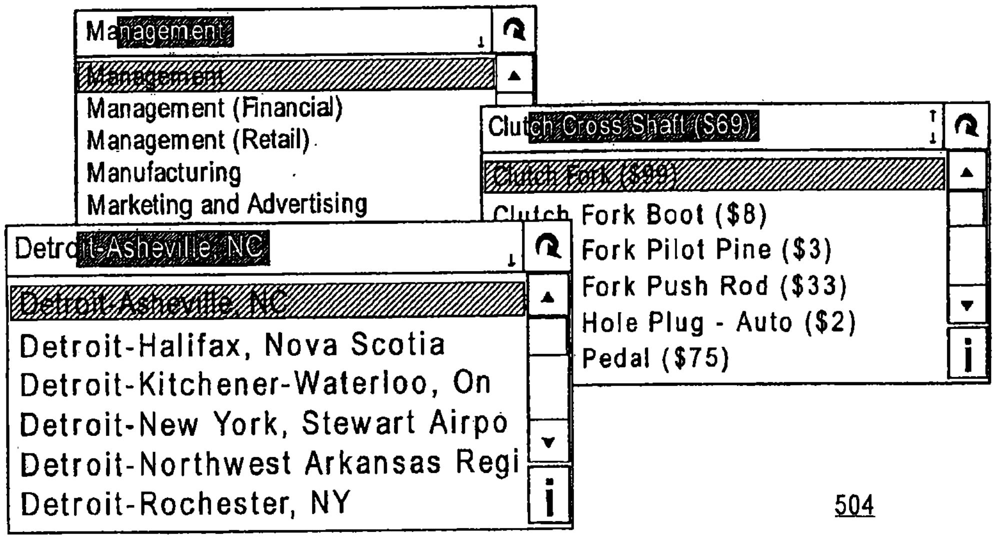
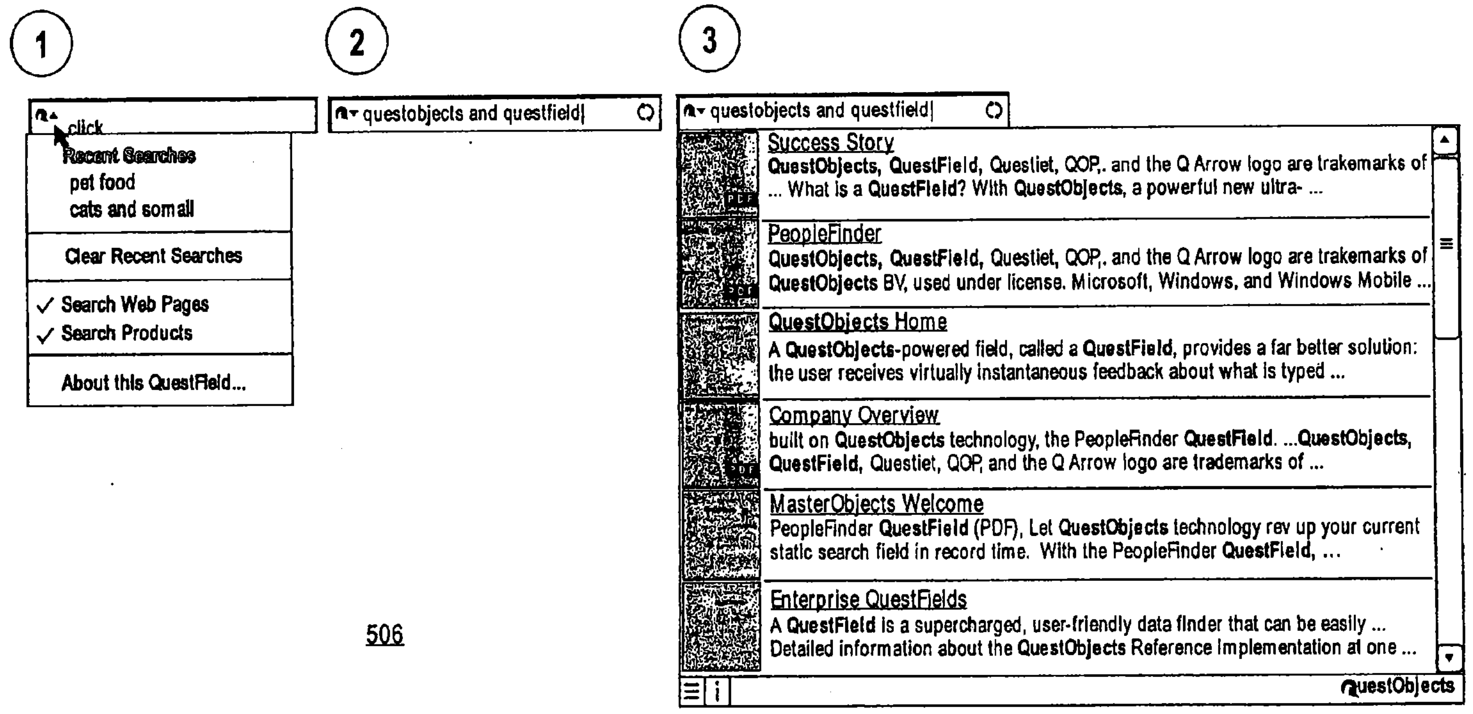

Management
Management
Management (Financial)
Management (Retail)
Manufacturing
Marketing and Advertising
Clutch Cross Shaft ($69)
Clutch Fork ($99)
Clutch Fork Boot ($8)
Fork Pilot Pine ($3)
Fork Push Rod ($33)
Hole Plug - Auto ($2)
Pedal ($75)
Detroit-Asheville, NC
Detroit-Asheville, NC
Detroit-Halifax, Nova Scotia
Detroit-Kitchener-Waterloo, On
Detroit-New York, Stewart Airpo
Detroit-Northwest Arkansas Regi
Detroit-Rochester, NY
504

FIG. 33



Success Story
QuestObjects, **QuestField**, Questlet, QOP,. and the Q Arrow logo are trakemarks of ... What is a **QuestField**? With **QuestObjects**, a powerful new ultra- ...

PeopleFinder
QuestObjects, **QuestField**, Questlet, QOP,. and the Q Arrow logo are trakemarks of **QuestObjects** BV, used under license. Microsoft, Windows, and Windows Mobile ...

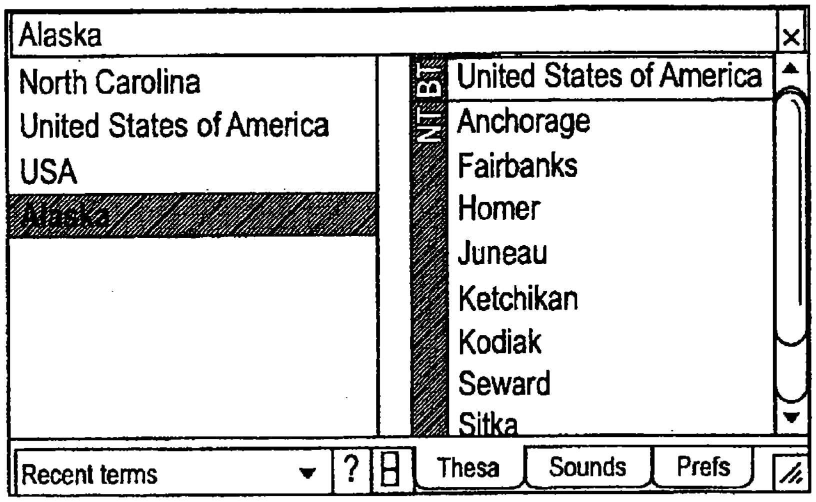
QuestObjects Home
A **QuestObjects**-powered field, called a **QuestField**, provides a far better solution: the user receives virtually instantaneous feedback about what is typed ...

Company Overview
built on **QuestObjects** technology, the PeopleFinder **QuestField**. ...**QuestObjects**, **QuestField**, Questlet, QOP, and the Q Arrow logo are trademarks of ...

MasterObjects Welcome
PeopleFinder **QuestField** (PDF), Let **QuestObjects** technology rev up your current static search field in record time. With the PeopleFinder **QuestField**, ...

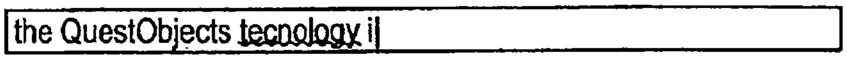
Enterprise QuestFields
A **QuestField** is a supercharged, user-friendly data finder that can be easily ... Detailed information about the **QuestObjects** Reference Implementation at one ...

FIG. 34



508

FIG. 35



510

FIG. 36

QUESTFIELD PROPERTIES						
QuestField Type	AutoLookup QuestField	AutoLookup QuestField	AutoSearch QuestField	Relational QuestField	FreeForm QuestField	Background QuestField
Relationship to data	One-to-one	One-to-many	Unrelated	Many-to-many	Many-to-many	Any
Number of results	1	Known	Unknown	Known	Known	Any
Listing	None	Alphabetized	Ranked	Custom	Optional	NA
Autocomplete	Optional	Full input	Last word (Optional)	Optional	Last word (Optional)	NA
Popup list	No	Yes	Yes	Optional	Optional	NA
Inputs	1	2	1	Multiple	1	Programmable
Content channels	1	1	1	Multiple	1 or more	1
Simultaneous queries	1	1	1	Multiple	Multiple	Any
Dependency target	Yes	Yes	Yes	Optional	No	Programmable

512

FIG. 37

SYSTEM AND METHOD FOR UTILIZING ASYNCHRONOUS CLIENT SERVER COMMUNICATION OBJECTS

CLAIM OF PRIORITY

[0001] This application is a continuation-in-part of U.S. patent application Ser. No. 09/933,493, entitled "SYSTEM AND METHOD FOR A SYNCHRONOUS CLIENT SERVER SESSION COMMUNICATION", filed Aug. 20, 2001; and also claims the benefit of U.S. Provisional Patent Application Ser. No. 60/622,907, entitled "SYSTEM AND METHOD FOR UTILIZING ASYNCHRONOUS CLIENT SERVER COMMUNICATION OBJECTS", filed Oct. 28, 2004; both of which applications are incorporated herein by reference.

COPYRIGHT NOTICE

FIELD OF THE INVENTION

[0003] The invention relates generally to client-server communication systems, and particularly to systems and methods for utilizing asynchronous client server communication objects for interactive database searching, data entry, online purchasing, and other applications.

BACKGROUND

[0004] The world has moved to embrace the Internet. In fact, the Internet has become a world of its own: a world of information, a world of marketing, a world in which virtually anything can be brought to anyone, anywhere. If the Internet is a world in itself, then intranets are enterprises in themselves. Knowledge and domain components—from purchasing orders, insurance policies and tax returns to shoes, chainsaws and paper clips, as well as customers, employees, and infrastructure—everything is available through Web applications, or soon will be. The Internet offers an enormous connectivity advantage: the ability to maintain information and business rules in one place, accessible by anyone we wish it to be accessible to.

[0005] Like most modern software applications, browser-based applications typically use a client that runs in a web browser or on a handheld wireless device, a server that provides a centralized application that centrally manages application data and the business logic, and a protocol that governs the communications between the client and the server. However, applications designed for the Internet are far more primitive and far less powerful than LAN-based client/server applications because web browsers must work on as many platforms and systems as possible, and because the protocols that made the Internet a worldwide success were, by design, limited in features. As such, missing from Web applications are manageable windows, multiple document interfaces, drag-and-drop, in-line editing, automatic completions, different views on the same data, updating while typing or selecting, automatic spell checking, intelligent lookups, instant calculations, and many other powerful interactive end user tools that are now standard features on personal computers and workstations everywhere. Typically, users have to press Submit, Search, Continue, Next, or a similar button for their input to have any effect, often resulting in a completely new page loading into their browser. As Web applications attempt to handle increasingly complex business data, users find themselves filling in huge forms, then being notified about typos or incompleteness only after pressing Submit.

[0006] To address the deficiencies and limitations of the web browser, a new class of client technologies has emerged. These technologies enable developers to create "rich" Internet applications (RIAs). RIAs are friendly, data-driven applications that run in web browsers and other "thin" client environments, providing advanced UI features that significantly enhance the browser user's experience.

[0007] RIAs can be developed using frameworks and technologies based on several popular platforms, including Macromedia Flash, Sun Java, Microsoft ASP. NET, DHTML (Dynamic HTML), JavaScript, HTML, Extensible Markup Language (XML), cascading style sheets (CSS), the Document Object Model (DOM), and the Microsoft XMLHTTPRequest object.

[0008] What, then, are users missing, from an interaction perspective, in current Web pages? It is not the mouse, which is an intrinsic part of any Web experience and is often the only device available to interact with a Web page. It is also not the WYSIWYG nature of modern GUI-based applications. HTML in itself is quite rich in formatting text, adding pictures, movies and everything else that make web pages appealing. HTML is designed primarily for the presentation of such information. However, because it must display this information using different browsers across a diverse range of platforms and systems, HTML offers only a simple user interface that is relatively primitive by modern server-based application standards. Consequently, Web applications are seriously handicapped when delivering interactive applications. Alternatives, such as replacing HTML pages by Java applets or full-page Flash applications, can enhance interaction but they have other disadvantages, which is a reason many companies tend to stick with HTML and JavaScript when moving their applications to the Web.

[0009] What users are primarily missing from their Internet or online environment is feedback. Web applications cannot provide users with feedback, the essential element of intelligent interaction that users have come to expect from their personal computers and workstations. Web pages are relatively static. They cannot be automatically responsive to user input. Instead, users must push a Submit button and wait for the page to redraw before getting any useful feedback on the data they typed into a form. This is totally contrary to the user experience provided by today's stand-alone or client/server applications (e.g. Microsoft Word) where instantaneous feedback is a common and expected feature (e.g. highlighting of spelling mistakes). The ability to provide immediate feedback on user actions would be advantageous in turning Web applications into interactive applications.

SUMMARY

[0010] As described herein, the QuestObjects system and method is a technology that adds interaction to Web applications. Working hand-in-hand with HTML, the technology allows Web servers to act on string input on a per character basis, thus enabling intelligent auto completion and complex lookups using server side data. In short, the system enables interactive data-driven behavior based on incremental string input. For Web applications, this offers: Improved data entry speed and accuracy; Dramatically faster access to relevant data; Improved data security; and Improved user friendliness. In some implementations the system may also be used to provide dynamically focused suggestions to the user.

[0011] By enhancing rather than replacing HTML, the system enables server interaction and improves interface usability and responsiveness without changing the nature or look-and-feel of Web applications. This architecture allows services to easily and transparently blend in with current Web applications. Moreover, clients work with the vast majority of Internet browsers now in use—with no additional software for the end user to install. Additionally, as a standards-based Web service, the system functions seamlessly with modern Java-based, .Net, and other architectures, and imposes no restrictions on networks or firewalls.

BRIEF DESCRIPTION OF THE FIGURES

[0012] **FIG. 1** shows an illustration of an example of a QuestObjects system and architecture in accordance with an embodiment of the invention.

[0013] **FIG. 2** shows an illustration of a system in accordance with an embodiment of the invention.

[0014] **FIG. 3** shows an illustration of a system in accordance with an embodiment of the invention.

[0015] **FIG. 4** shows an illustration of an asynchronous session-based search system including a front-end client search field and a back-end server datastore.

[0016] **FIG. 5** shows an illustration of an asynchronous session-based search system including multiple front-end client search fields, multiple channels, and a back-end server datastore.

[0017] **FIG. 6** shows an illustration of an asynchronous session-based search system including multiple front-end client search fields.

[0018] **FIG. 7** shows an illustration of an asynchronous session-based search system including a front-end client search fields, a server, result storage, and a back-end server datastore.

[0019] **FIG. 8** shows an illustration of a multi-tier asynchronous session-based search system including client tier, server tier, service tier, and content tier.

[0020] **FIG. 9** shows an illustration of an asynchronous session-based search system for use with web forms or other web interfaces.

[0021] **FIG. 10** shows an illustration of a web interface in accordance with the prior art.

[0022] **FIG. 11** shows an illustration of a web-based search field in accordance with an embodiment of the invention.

[0023] **FIG. 12** shows a listing of a html and JavaScript code in accordance with an embodiment of the invention.

[0024] **FIG. 13** shows an illustration of a web-based search field as it is used to receive data from a server in accordance with an embodiment of the invention.

[0025] **FIG. 14** shows a screenshot of a music search input screen in accordance with the prior art.

[0026] **FIG. 15** shows a screenshot of a music search input screen in accordance with the prior art.

[0027] **FIG. 16** shows a screenshot of a music record search input screen in accordance with an embodiment of the invention.

[0028] **FIG. 17** shows a screenshot of a music record search input screen in accordance with an embodiment of the invention.

[0029] **FIG. 18** shows a screenshot of a music record search input screen in accordance with an embodiment of the invention.

[0030] **FIG. 19** shows a screenshot of a music record search input screen in accordance with an embodiment of the invention.

[0031] **FIG. 20** shows a screenshot of a music record search input screen in accordance with an embodiment of the invention.

[0032] **FIG. 21** shows a screenshot of a music record search input screen in accordance with an embodiment of the invention.

[0033] **FIG. 22** shows a screenshot of a music record search input screen in accordance with an embodiment of the invention.

[0034] **FIG. 23** shows a screenshot of a music record search input screen in accordance with an embodiment of the invention.

[0035] **FIG. 24** shows a screenshot of a person search input screen in accordance with an embodiment of the invention.

[0036] **FIG. 25** shows a screenshot of a person search input screen in accordance with an embodiment of the invention.

[0037] **FIG. 26** shows a screenshot of a person search input screen in accordance with an embodiment of the invention.

[0038] **FIG. 27** shows a screenshot of a person search input screen in accordance with an embodiment of the invention.

[0039] **FIG. 28** shows a screenshot of a multiple field search input screen in accordance with an embodiment of the invention.

[0040] **FIG. 29** shows a screenshot of an alternate person search input screen in accordance with an embodiment of the invention.

[0041] **FIG. 30** shows a screenshot of an alternate person search input screen in accordance with an embodiment of the invention.

[0042] **FIG. 31** shows a screenshot of an alternate person search input screen in accordance with an embodiment of the invention.

[0043] FIG. 32 shows an illustration of a QuestFields type in accordance with an embodiment of the invention.

[0044] FIG. 33 shows an illustration of a QuestFields type in accordance with an embodiment of the invention.

[0045] FIG. 34 shows an illustration of a QuestFields type in accordance with an embodiment of the invention.

[0046] FIG. 35 shows an illustration of a QuestFields type in accordance with an embodiment of the invention.

[0047] FIG. 36 shows an illustration of a QuestFields type in accordance with an embodiment of the invention.

[0048] FIG. 37 shows a table comparing different QuestField types in accordance with an embodiment of the invention.

DETAILED DESCRIPTION

[0049] As described herein, the QuestObjects system and method is a technology that adds interaction to Web applications. Working hand-in-hand with HTML, the technology allows Web servers to act on string input on a per character basis, thus enabling intelligent auto completion and complex lookups using server side data. In short, the system enables interactive data-driven behavior based on incremental string input. For Web applications, this offers: Improved data entry speed and accuracy; Dramatically faster access to relevant data; Improved data security; and Improved user friendliness. In some implementations the system may also be used to provide dynamically focused suggestions to the user.

[0050] By enhancing rather than replacing HTML, the system enables server interaction and improves interface usability and responsiveness without changing the nature or look-and-feel of Web applications. This architecture allows services to easily and transparently blend in with current Web applications. Moreover, clients work with the vast majority of Internet browsers now in use—with no additional software for the end user to install. Additionally, as a standards-based Web service, the system functions seamlessly with modern Java-based, .Net, and other architectures, and imposes no restrictions on networks or firewalls.

[0051] The system offers a highly effective solution to the aforementioned disadvantages of both client-server and Internet systems by providing a way to synchronize the data entered or displayed on a client system with the data on a server system. Data input by the client can be immediately transmitted to the server, at which time the server can immediately update the client display. To ensure scalability, systems built around the QuestObjects concept can be divided into multiple tiers, each tier being capable of caching data input and output. A plurality of servers can be used as a middle-tier to serve a large number of static or dynamic data sources, herein referred to as "content engines."

[0052] A variety of embodiments may be designed to suit a correspondingly wide variety of applications. As such the system offers a standardized way to access server data that allows immediate user-friendly data feedback based on user input. Data can also be presented to a client without user input, i.e. the data are automatically pushed to the client. This enables a client component to display the data immediately, or to transmit the data to another software program to be handled as required.

[0053] The system can also be used to simply and quickly retrieve up-to-date information from any string-based content source. Strings can be linked to metadata allowing user interface components to display corresponding information such as, for example, the meaning of dictionary words, the description of encyclopedia entries or pictures corresponding to a list of names.

[0054] Embodiments of the system can be used to create a user interface component that provides a sophisticated "auto-completion" or "type-ahead" function that is extremely useful when filling out forms. This is analogous to simple, client-side auto-complete functions that have been widely used throughout the computing world for many years. As a user inputs data into a field on a form, the auto-complete function analyzes the developing character string and makes intelligent suggestions about the intended data being provided. These suggestions change dynamically as the user types additional characters in the string. At any time, the user may stop typing characters and select the appropriate suggestion to auto-complete the field.

[0055] Today's client-side auto-complete functions are useful but very limited. The system, however, in its various embodiments, vastly expands the usefulness and capabilities of the auto-complete function by enabling the auto-complete data, logic and intelligence to reside on the server, thus taking advantage of server-side power. Unlike the client-side auto-complete functions in current use, an auto-complete function created by the system generates suggestions at the server as the user types in a character string. The suggestions may be buffered on a middle tier so that access to the content engine is minimized and speed is optimized.

[0056] The simple auto-complete schemes currently in popular use (such as email programs that auto-complete e-mail addresses, web browsers that auto-complete URLs, and cell phones that auto-complete names and telephone numbers) require that the data used to generate the suggestions be stored on the client. This substantially limits the flexibility, power, and speed of these schemes. Embodiments of the system, however, store and retrieve the auto-complete suggestions from databases on the server. Using the system, the suggestions generated by the server may, at the option of the application developer, be cached on the middle tier or on the client itself to maximize performance.

[0057] The system provides better protection of valuable data than traditional methods, because the data is not present on the client until the moment it is needed, and can be further protected with the use of user authentication, if necessary.

[0058] The system is also useful in those situations that require immediate data access, since no history of use needs to be built on the client before data is available. Indeed, data entered into an application by a user can automatically be made available to that user for auto-completion on any other computer, anywhere in the world.

[0059] Unlike existing data-retrieval applications, server data can be accessed through a single standardized protocol that can be built into programming languages, user interface components or web components. The system can be integrated into and combined with existing applications that access server data. Using content access modules, the system can access any type of content on any server.

[0060] In the detailed description below, embodiments of the present invention are described with reference to a particular embodiment named QuestObjects, created by the MasterObjects company. QuestObjects provides a system and method for managing client input, server queries, server responses and client output. One specific type of data that can be made available through the system from a single source (or syndicate of sources) is a QuestObjects Service. Other terms used to describe the QuestObjects system can be found in the glossary given below. It will be evident that the technology described herein may be utilized in other embodiments, and with other systems, in addition to the QuestObjects system.

[0061] **FIG. 1** shows an example of the QuestObjects architecture **100**. Generally described, QuestObjects is a powerful ultra-thin smart client/server technology used to create intelligent online data entry and retrieval applications called QuestFields. QuestFields, the products based on the QuestObjects technology, are deployed in web browser and handheld wireless device applications and enable up to millions of simultaneous users to have direct, virtually instantaneous access to enterprise data on remote content sources.

[0062] QuestFields compete primarily in the RIA market. However, unlike competitive products, a QuestField is an integrated "end-to-end" client-server solution that is more powerful, more universal and easier to deploy—all at a substantially lower cost than typical RIAs.

[0063] QuestFields are comprised of three integrated and highly optimized parts: the QuestObjects client, the QuestObjects Server, and the QuestObjects Protocol (QOP). The different parts of QuestFields can be distributed over multiple computers to provide load balancing and optimal performance.

[0064] The QuestObjects client typically comprises one or more QuestFields that enable each user to efficiently query remote content sources by providing a friendly but powerful user interface that communicates directly with the QuestObjects Server over the Internet. The QuestObjects Server easily handles many simultaneous user sessions and provides the interface to the underlying content sources (such as databases, directories, or search engines). The QuestObjects Server enables administrators to easily configure any number of content channels, each of which queries one of potentially many content sources that are typically present on a remote local area network (LAN).

Advantages of the QuestFields

[0065] Compared to existing RIA client technologies, QuestObjects-based products offer several important advantages, including: Far better performance; Proven functionality in large-scale corporate environments; No rewriting of existing web application code or redesign of web page layout; Compatibility with more than 99% of web browsers currently in use; Substantially faster implementation time; and Substantially lower implementation and maintenance cost.

[0066] QuestFields are designed to be compatible with virtually all "thin" client platforms. QuestField products can be designed to look like a "combo box" input field, that are used primarily in web browsers. However, future QuestFields can come in many more shapes, sizes, types and uses. Unlike other RIA technologies, QuestFields are truly universal because they can be developed in virtually any programming language that is supported by web browsers. Consequently, QuestFields will always be able to take advantage of the best available client technologies, even those yet to be developed.

Advantages of the QuestObjects Server

[0067] The QuestObjects Server competes with custom web application development environments, dedicated Web Services, groupware, and connectivity products that also provide a means to access content sources from within the browser. The QuestObjects Server has several advantages over competing server products, including: Easily configurable "black box" application requiring no programming and virtually no maintenance; Enables users to retrieve information from virtually any content source without the need to develop and maintain a custom application or Web Service; Provides a highly optimized service that enables many simultaneous users to access content with minimal impact on the customer's network or content engines; Usable in a far broader market than other groupware and connectivity products that typically replace full applications; Significantly enhances other, more complex, web applications, groupware, and connectivity products; and Runs 24/7, automatically connects to redundant content sources, and requires virtually no systems management.

Advantages of the QuestObjects Protocol

[0068] The QuestField and the QuestObjects Server communicate with each other using the QuestObjects Protocol (QOP). QOP is a standards-compliant communications protocol fully compatible with Service Oriented Architectures (SOA). SOA is an architectural approach that segments and isolates application functionality into smaller, discrete and usable components, otherwise known as "services." The primary goal of a SOA is developing application functions that are reusable and standardized so that once created they can be leveraged across multiple projects. This approach greatly reduces time, effort and cost of incorporating new functionality and extending existing applications. The QuestObjects technology enables organizations to do precisely that and to do it simply, quickly, and easily.

[0069] QOP uses the same transport mechanism that is used by standard web pages: HTTP over TCP/IP. As with all modern SOAs, this allows QOP to transparently communicate over the Internet without imposing unusual requirements on routers and firewalls. By contrast, legacy application protocols typically rely on dedicated ports and required specialized drivers to be installed on client and server.

QuestObjects Server

[0070] In accordance with an embodiment, the QuestObjects Server is an application that runs in a standard Java Servlet Container, compatible with open-source and commercial Java application servers that are used in enterprises throughout the world. A QuestObjects Server provides its QuestObjects Services through content channels. Each content channel returns a specific kind of data from a specific back-end content source. For handling different kinds of back-end data, the QuestObjects Server enables the use of multiple Content Access Modules (CAMs) that each provide a means to communicate with a specific kind of content engine on the content source, such as SQL, LDAP, or a proprietary legacy database. QuestObjects Server features include:

[0071] Request Management—The QuestObjects Server manages the load of incoming client requests and queries to the content source. Request management enables the server to scale to very large numbers of users and queries.

[0072] User Session Management—The QuestObjects Server provides efficient metering and auditing by using the data in each user's session to keep track of the queries a user performs and the results that have been sent back.

[0073] Unified Query Cache—The QuestObjects Server caches query results in a cache that is common to all users, thus improving performance on recurring queries and limiting the load imposed on content engines.

[0074] Unlimited Content Sources—The QuestObjects Server can query one or more content sources. Support for both SQL databases and LDAP directories is built-in, and a modular Java interface provides a simple API that enables MasterObjects and its customers to easily and quickly create custom interfaces to legacy data.

[0075] Query Merging—Without requiring any additional programming, a QuestObjects content channel can perform one or more queries on the back-end database or directory and intelligently combine their results into a single results list. This makes it very easy to implement QuestFields that enable users to perform queries in alternate ways, such as looking up a person by first name, last name, email address, or any combination thereof. A single content channel can even combine results from multiple different content sources.

QuestObjects Protocol (QOP)

[0076] In accordance with an embodiment, to enable the QuestObjects technology to communicate efficiently over the Internet, a protocol, called the QuestObjects Protocol (QOP), is used for communication between large numbers of simultaneous QuestField users and any number of QuestObjects Servers. QOP uses the very same network infrastructure that is used by standard web pages. This means that if a web page loads correctly into the browser, QOP works as well. Consequently, neither users nor administrators need to worry about the details of the communication protocol. In accordance with an embodiment, QOP client-server messaging is based on web standards. The application-layer protocol is based on XML wrapped in optional SOAP envelopes using HTTP(S) as the transport layer. QOP does not require the use of cookies in the browser and is designed to be compliant with existing Internet and security standards.

Security

[0077] QOP can be configured to run over Secure Sockets Layer (SSL) for complete security of user queries and data received from the server. Either the entire web page or individual QuestField queries can be securely encrypted. This means that a web page using QuestFields can load very quickly by keeping its images unencrypted, yet still fully securing content that appears in its QuestFields.

Load Balancing

[0078] The QuestObjects technology is specifically designed for large intranet and Internet applications. The QuestObjects Server, QuestObjects Service and/or the content source can reside simultaneously on multiple machines, permitting load balancing and capacity expansion simply by adding more hardware. A QuestObjects Server uses "sticky" session connections so that a client can logically connect to any server machine in the system. Once a session is established, all communications from the client IP address go to and from the same server.

QuestObjects Services

[0079] In accordance with an embodiment, each QuestObjects Server can be configured to provide QuestObjects Services that are available to users of independent websites. This makes virtually any content that is available on the Internet accessible to QuestFields. QuestObjects Services can be provided from Internet domains other than the domain that serves the web pages. Thus, QuestField users can subscribe to multiple QuestObjects Services that are hosted by different providers on the Internet. To manage these services, the QuestObjects technology uses Syndicators, which offer content provided through QuestObjects Servers. Syndicators offer subscription-based access to specific content channels to managed user groups, enabling the QuestObjects Server to automatically collect usage statistics and provide billing information for commercial use of the service.

Glossary

AutoComplete QuestField

[0080] A type of QuestField that closely resembles the "combo box" of traditional applications, whereby user entry is automatically completed, and multiple results can be displayed in a popup list.

AutoLookup QuestField

[0081] The simplest type of QuestField, which performs a direct lookup based on the user's input and displays the corresponding single result.

AutoSearch QuestField

[0082] A type of QuestField that is used to enter user queries (such as Boolean) and to display corresponding results in a filtered and ranked result list.

Background QuestField

[0083] A type of QuestField that has no user interface, but rather is integrated into an application where it runs in the background accessing data from QuestObjects Services.

CAM

[0084] See Content Access Module.

CSS

[0085] Cascading Style Sheets (CSS) is a style sheet language used to describe the presentation of a document written in HTML (or other markup languages). It allows the "look" of a web page to be modified without changing the underlying HTML or web application, and thus separates the "presentation logic" from the "application logic" and business rules. An embodiment of the invention takes advantage of CSS to enable customers to change the "look" of QuestFields so they "blend in" to their own web pages. Customers can easily modify the colors and widths of QuestField borders, as well as the images used for QuestField buttons.

Content Access Module

[0086] A Content Access Module (CAM) provides a standardized mechanism to link the QuestObjects system to a specific type of content engine. A CAM is the "middleware" between the QuestObjects system and the data it accesses. QuestObjects currently includes CAMs that communicate with any JDBC-compliant database or any LDAP-compliant directory server, as well as a Java CAM that allows customers to easily integrate the QuestObjects Server with their legacy or proprietary databases by using a powerful yet simple open Java API.

Content Channel

[0087] A configuration on the QuestObjects Server that defines a specific method of querying one or more specific content sources, allowing QuestField users to perform queries and retrieve corresponding results. A content channel accesses one or more content engines, each through a specific Content Access Module. A single content channel can be configured to perform multiple queries to retrieve data from the content sources, whereby the QuestObjects Server merges the results from these "sub queries" into a single result set for the QuestField user. For example, in a PeopleFinder application the content channel can be configured to query the underlying content engine by last name, first name, email address, and any combination thereof. The QuestField user receives a consolidated list of person names that were returned by any of the sub queries in the content channel.

Content Engine

[0088] A content engine is a third-party application that runs on the content source that is capable of performing string-based queries and returning string-formatted answers to the QuestObjects system. Examples include relational databases, corporate directories, and search engines. A simple content engine could read information directly from a file, or could perform a query to access a Web Service over the Internet. The QuestObjects Server simultaneously accesses different content engine types through Content Access Modules.

Content Source

[0089] A server computer that provides the data that is accessed by the QuestObjects system. The content source makes its data available through a content engine. For best performance, the content source must be located on the same LAN as the QuestObjects Server, and could even be hosted on the very same server computer. The QuestObjects Server can be linked to any number of content sources. To retrieve specific information from the content source, one or more content channels are configured on the QuestObjects Server.

DHTML QuestField

[0090] A version of QuestField that is based on DHTML technology. DHTML enables the QuestField to run in modern web browsers without requiring Flash or other plug-in technology. QuestObjects technology enables QuestFields to detect the browser, and to automatically activate the appropriate QuestField for each individual user. Future QuestFields may be built using alternative client technologies, such as J2ME.

ECMAScript

[0091] See JavaScript.

Flash

[0092] Multimedia authoring program and a corresponding runtime environment called the Macromedia Flash Player, written and distributed by Macromedia, that utilizes vector and raster graphics, program code and bidirectional streaming video and audio. Strictly speaking, Macromedia Flash is the authoring environment and Flash Player is the virtual machine application used to run the Flash files, but in colloquial language these have become mixed: "Flash" can mean either the authoring environment, the player or the application files. The Flash files, which usually have an SWF file extension, may appear in a web page for viewing in a web browser, or standalone Flash players may "play" them. Flash files occur most often as animations or design elements on web pages, and more recently Rich Internet Applications. They are also widely used in web advertisements, due to the fact that a flash file can contain much more information than a GIF or JPEG file of the same size.

Flash QuestField

[0093] A version of QuestField that is based on Flash technology, allowing it to run in any browser that has the Flash Player installed.

FreeForm QuestField

[0094] A type of QuestField that consists of a large text area and provides data management services such as remote spell checking and auto-save.

HTTP

[0095] HTTP (Hypertext Transfer Protocol) is the set of rules for transferring files (text, graphic images, sound, video, and other multimedia files) using the Internet protocol (TCP/IP) on the World Wide Web.

HTTPS

[0096] HTTPS (HTTP over SSL) is an extension to HTTP that provides security by encrypting and decrypting user page requests as well as the pages that are returned by the web server.

Java

[0097] Java is an object-oriented programming language developed by Sun Microsystems. Specifications of the Java language, the JVM (Java Virtual Machine) and the Java API are community-maintained through the Sun-managed Java Community Process.

Applets

[0098] Small applications written in Java that run in any web browser that supports a JVM (Java Virtual Machine). Client-side Java applications and Java applets have never become a predominant client technology.

JavaScript

[0099] Object-based scripting programming language that is built into web browsers, also known as ECMAScript after the standards body that now governs the language. JavaScript is best known for its use in websites, but is also used to enable scripting access to objects embedded in other applications.

Java Servlet Container

[0100] Part of Java application servers such as IBM WebSphere, BEA WebLogic, and Apache Tomcat that allows multiple Servlet-based applications to be hosted on a web server. A servlet container controls the servlets that are deployed within the web server, and is responsible for forwarding the requests and responses for them. It has the functionality of mapping a URL to a particular servlet and of ensuring that the process requesting the URL has the correct access rights.

JDBC

[0101] Java Database Connectivity, or JDBC, is an API for the Java programming language that defines how a client may access a database. It provides methods for querying and updating data in a database. JDBC is oriented towards relational databases that use SQL. An embodiment of the invention includes a Content Access Module that allows QuestObjects to query databases through JDBC. JDBC is supported by virtually all commercial and open-source SQL databases including Oracle, IBM DB2, Microsoft SQL Server, MySQL, etc.

J2ME

[0102] Acronym for Java 2 Platform, Micro Edition (recently renamed by Sun to Java ME, but still most often referred to as J2ME), a collection of Java APIs targeting embedded products such as PDAs, cell phones and other consumer appliances. J2ME has become a popular option for creating games for cell phones, as they can be emulated on a PC during the development stage and easily uploaded to the phone.

JVM

[0103] Acronym for Java Virtual Machine. All applications that were built in Java run in a JVM, which is available for virtually all operating systems and embedded devices.

LDAP

[0104] The Lightweight Directory Access Protocol, or LDAP, defines a relatively simple and efficient protocol for updating and searching directories running over the Internet protocol, TCP/IP. It is in common use in enterprises worldwide. Virtually all commercial and open-source directory servers use LDAP, allowing applications to access directory information in a standardized way, similar to the way in which JDBC provides a way to access SQL databases. An embodiment of the invention includes a Content Access Module that allows QuestObjects to query directories through LDAP.

PeopleFinder QuestField

[0105] Product name used for a specific kind of AutoComplete QuestField: Configured to access a content channel that retrieves personnel records, enabling users to use their web browser to quickly find names, addresses, phone numbers, etc. in a corporate directory or personnel database without leaving the HTML page they are on.

QOP

[0106] See QuestObjects Protocol.

QuestField

[0107] A user interface element in a browser-based "Rich Internet Application" that sends queries to, and receives results from the QuestObjects Server. Forms the client part of the QuestObjects technology. Six different types of QuestFields are envisioned: AutoLookup, AutoComplete, AutoSearch, Relational, FreeForm, and Background

QuestObjects Enterprise Server

[0108] Commercial name for the QuestObjects Server product that is optimized and licensed for use in a closed intranet setting, where the total number of users is known.

QuestObjects Protocol

[0109] The QuestField and the QuestObjects Server communicate with each other using the QuestObjects Protocol (QOP). QOP is a standards-compliant communications protocol that fits well into Service Oriented Architectures (SOA). It is based on small XML-formatted data packages that are exchanged over the Internet using HTTP.

QuestObjects Server

[0110] A Java server application that implements the server part of the QuestObjects product. It communicates with the QuestObjects Client (QuestFields) through the QuestObjects Protocol, and communicates with one or more content engines through Content Access Modules.

QuestObjects Service

[0111] A logical name for one or more content channels that provide a valuable service for QuestField users on the Internet. A QuestObjects-specific Web Service.

Relational QuestField

[0112] A type of QuestField that provides multiple inputs to the user allowing navigation through complex relational data structures configured in multiple content channels.

Service Oriented Architectures

[0113] SOA is an architectural approach that segments and isolates application functionality into smaller, discrete and usable components, otherwise known as Web Services. The primary goal of a SOA is developing application functions that are reusable and standardized so that once created they can be leveraged across multiple projects. This approach greatly reduces time, effort and cost of incorporating new functionality and extending existing applications. The QuestObjects technology enables organizations to do precisely that and to do it simply, quickly, and easily.

Servlet

[0114] A servlet is an object in a Java server application that receives requests and generates a response based on each request. The QuestObjects Server implements servlets to perform these basic yet essential tasks, leveraging standard Java Servlet Container technology for optimal performance and full compatibility with other server technologies in common use. See Java Servlet Container.

Site Search QuestField

[0115] An AutoSearch QuestField that is customized and optimized for performing web searches. Site Search QuestFields access a content channel that performs queries on the full-text index of one or more websites, as well as databases used by dynamic websites.

Site Search Service

[0116] A QuestObjects Service that is accessed by Site Search QuestFields.

SOA

[0117] See Service Oriented Architectures.

SOAP

[0118] Simple Object Access Protocol (SOAP) provides a standardized structure for XML-based information used for exchanging structured and typed information between peers in a decentralized, distributed environment. It is most commonly used to package XML-formatted data that is exchanged in Service Oriented Architectures.

SQL

[0119] Structured Query Language (SQL, often pronounced as "sequel") is the most popular computer language used to create, modify and retrieve data from relational database management systems. It is in common use in enterprises worldwide. Virtually all commercial and open-source databases use SQL, allowing applications to access database information in a standardized way, similar to the way in which LDAP provides away to access corporate directories. An embodiment of the invention includes a SQL Content Access Module that allows QuestObjects to query databases through JDBC.

SSL

[0120] The Secure Sockets Layer (SSL) is a commonly used protocol for managing the security of message transmission on the Internet.

Syndicator

[0121] A logical component in QuestObjects Server that manages a group of content channels for a group of users. A Syndicator manages user access privileges through subscriptions to one or more content channels, performs metering, and can be used as a source for billing information.

Web Service

[0122] A Web Service is a collection of protocols and standards used for exchanging data between applications or systems that implement a Service Oriented Architecture. Software applications written in various programming languages and running on various platforms can use Web Services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability is due to the use of open standards.

[0123] Embodiments of the present invention provide a system and a method that allows clients or client applications to asynchronously retrieve database information from a remote server or server application. The terms "client" and "server" are used herein to reflect specific embodiments of the invention, although it will be evident to one skilled in the art that the invention may be equally used with any implementation that requires communication between a first process or application and a second process or application, regardless of whether these processes comprise a typical client-server setup or not. The invention includes a Server, that handles requests for information from clients, and a communication protocol that is optimized for sending characters from a Client to the Server, and lists of strings from the Server to the Client. In one embodiment, as the Server receives a single character from the Client, it immediately analyzes the lengthening string of characters and, based on that analysis, returns database information to the Client in the form of a list of strings. Clients are not restricted to programs with a user interface. Generally, any process or mechanism that can send characters and receive string lists can be considered a client of the system. For example, in an industrial or power supply setting, the control system of a power plant could send sensor readings to the system, and in return receive lists of actions to be taken, based on those sensor readings.

[0124] The system's protocol is not restricted to sending single characters. In fact, Clients can also use the protocol to send a string of characters. For example, when a user replaces the contents of an entry field with a new string, the Client may then send the entire string all at once to the Server, instead of character by character.

[0125] In accordance with one embodiment of the invention the system is session-based, in that the server knows or recognizes when subsequent requests originate at the same Client. Thus, in responding to a character the Server receives from a Client it can use the history of data that has been sent to and from the current user. In one embodiment, the system stores user preferences with each Service, so that they are always available to the Client, (i.e., they are independent of the physical location of the client). Furthermore, client authentication and a billing system based on actual data and content use by Clients are supported. For faster response, the Server may predict input from the Client based on statistics and/or algorithms.

[0126] The system is bi-directional and asynchronous, in that both the Client and the Server can initiate communications at any moment in time. The functionality of the system is such that it can run in parallel with the normal operation of clients. Tasks that clients execute on the system are non-blocking, and clients may resume normal operation while the system is performing those tasks. For example, a communication initiated by the Client may be a single character that is sent to the Server, that responds by returning appropriate data. An example of a communication initiated by the Server is updating the information provided to the client. Because the system is session-based it can keep track of database information that has been sent to the Client. As information changes in the database, the Server sends an updated version of that information to the Client.

[0127] Embodiments of the system may be implemented as a multi-tier environment This makes it scalable because the individual tiers can be replicated as many times as necessary, while load balancing algorithms (including but not limited to random and round robin load-balancing) can be used to distribute the load over the copies of the tiers. One skilled in the art would appreciate that it is not necessary to replicate the tiers. Indeed, there may be only a single copy of each tier, and that all tiers (Client, Server, and Service) may be running on a single computer system.

[0128] **FIG. 2** illustrates one example of a system that embodies the present invention. As shown in **FIG. 2** there may be various Clients 102 using the system. These Clients use a communication protocol 104 to send information, including but not limited to single characters, and to receive information, including but not limited to lists of strings and corresponding metadata. At least one Server 106 receives information from the Client, and sends information to the Client. In a typical embodiment if there is a plurality of Servers, then the system can be designed so that each Client connects to only one of them, which then relays connections to other Servers, possibly using load-balancing algorithms. Servers have a communication link **108** to a Service **110**, which they use to obtain the information that they send to the Client.

[0129] **FIG. 3** is a schematic illustrating an embodiment of the present invention, and displays a five-tier system that has a user interface in which user interface elements use the present invention to assist the user in performing its tasks. For purposes of illustration, **FIG. 3** displays just one session and one content Service. In an actual implementation there may be multiple concurrently active sessions, and there may be more than one content Service that Clients can use. As shown herein, the first of the five tiers is a Client tier **120**. The Client tier contains the user interface and the Client components that are needed to use the system. The second tier is a Server or server process **130**, which handles the queries that Clients execute, and in return displays results to the Client. Service **150**, which corresponds to **110** of **FIG. 2**, is a logical entity comprising three more tiers: a Syndicator **152**, a Content Channel **162** and a Content Engine **174**. The Syndicator provides access to a number of Content Channels and performs accounting services based on actual data use. The Content Channel provides a specific type of information from a specific source (i.e. the Content Engine). The Content Engine is the actual source of any content that is made available through the QuestObjects system. The Client tier **120** corresponds to the client **102** in **FIG. 2**. In this example, the Client may be an application (and in some embodiments a web application) with a user interface that accesses the system of the present invention. As used in the context of this disclosure a user interface element that uses the present invention is referred to as a "Questlet." A Client can contain one or more Questlets **122** (e.g. an input field or a drop down list. A Questlet is always associated with at least one Client Quester **124**. Questers are objects that tie a QuestObjects input buffer (containing input from the Client) to a QuestObjects Result Set returned from a QuestObjects Server. Questers exist on both the Client and Server, in which case they are referred to as a Client Quester and a Server Quester, respectively. Every Client Quester has one corresponding Server Quester. In accordance with the invention, any event or change that happens in either one of them is automatically duplicated to the other so that their states are always equal. This synchronization mechanism is fault-tolerant so that a failure in the communication link does not prevent the Questers from performing tasks for which they do not need to communicate. For example, a Client Quester can retrieve results from the cache, even if there is no communication link to the Server. Each single Quester accesses exactly one QuestObjects Service, i.e. one specific Content Channel offered by one specific Syndicator. At initialization of the Client, the Questlet tells its Quester which Service to access. In one embodiment a Service is stored or made available on only one Server within a network of Servers. However, this is transparent to the Client because each Server will forward requests to the right computer if necessary. The Client does not need to know the exact location of the Service.

[0130] To communicate with its Server Quester **134**, each Quester in a session uses a controller **126**. The system contains at least one Client Controller **126** and a Server Controller **136**, which together implement the network communication protocol **128** of the present invention. Client Controllers may cache results received from a Server, thus eliminating the need for network traffic when results are reused.

[0131] Client Questers are managed by a Questlet, which create and destroy Questers they need. In a similar fashion, ServerQuesters are managed by a Session **132**. When a Client Quester is created, it registers itself with the Client Controller. The Client controller forwards this registration information as a message to the Session using the Server Controller. The Session then checks if the Persistent Quester Store **138** contains a stored Quester belonging to the current user matching the requested Service and Query Qualifier. If such a Quester exists, it is restored from the Persistent Quester Store and used as the peer of the Client Quester. Otherwise, the Session creates a new Server Quester to be used as the Client Quester's peer.

[0132] A Time Server **140** provides a single source of timing information within the system. This is necessary, because the system itself may comprise multiple independent computer systems that may be set to a different time. Using a single-time source allows, for example, the expiration time of a Result Set to be calibrated to the Time Server so that all parts of the system determine validity of its data using the same time.

[0133] Server communication link **144** is used by the Server to send requests for information to a Service, and by a Service to return requested information. Requests for information are Query objects that are sent to and interpreted by a specific Service. Query objects contain at least a string used by the Service as a criterion for information to be retrieved, in addition to a specification of row numbers to be returned to the Client. For example, two subsequent queries may request "row numbers 1 through 5", and "6 through 10", respectively. A query object may also contain a Qualifier that is passed to the appropriate Service. This optional Qualifier contains attributes that are needed by the Service to execute the Query. Qualifier attributes may indicate a desired sort order or in the example of a thesaurus Service may contain a parameter indicating that the result list must contain broader terms of the Query string. Services use the communication link to send lists of strings (with their attributes and metadata) to Servers. Server communication link **144** is also used by Server Questers to store and retrieve user preferences from a Syndicator's Preference Manager.

[0134] Questers use Services to obtain content. A Service is one of the Content Channels managed by a Syndicator. When a Quester is initialized, it is notified by its QuestField (Active Component) of the Service it must use. The Service may require authentication, which is why the Syndicator provides a User Manager **154**. If a Client allows the user to set preferences for the Service (or preferences needed by the QuestField), it may store those preferences using the Syndicator's Preference Manager **156**. The Server (i.e. Server Quester) only uses the Syndicator for authentication and preferences. To obtain content, it accesses the appropriate Content Channel directly. The Content Channel uses its Syndicator to store usage data that can be later used for accounting and billing purposes. Usage data is stored in a Usage Statistics Store **158**.

[0135] Content communication link **160** is used by Content Channels to send usage data to their Syndicator, and to retrieve user information from the Syndicator. The Content Channel is a layer between the QuestObjects System, and the actual content made available to the system by a Content Engine **174**. Each Content Channel has a corresponding Query Manager **164** that specifies the type of query that can be sent to the corresponding Content Engine, and defines the types of data that can be returned by the Content Channel.

[0136] Specification of query type comprises a set of Query Patterns and Query Filters that are used by the Server Quester to validate a string before the string is sent to the Content Channel as a QuestObjects Query. For example, a query type "URL" may allow the ServerQuester to check for the presence of a complete URL in the input string before the input string is sent to the Content Channel as a query. A query type "date" might check for the entry of a valid date before the query is forwarded to the Content Channel.

[0137] The Query Manager optionally defines the types of string data that can be returned to the Client by the Content Channel. Specific QuestFields at the Client can use this information to connect to Services that support specific types of data. Examples of string types include: simple terms, definitional terms, relational terms, quotes, simple numbers, compound numbers, dates, URLs, e-mail addresses, preformatted phone numbers, and specified XML formatted data etc.

[0138] The Query Manager **164** retrieves database information through a Content Access Module **166**. The Content Access Module is an abstraction layer between the Query Manager and a Content Engine. It is the only part of the system that knows how to access the Content Engine that is linked to the Content Channel. In this way, Query Managers can use a standardized API to access any Content Engine. To reduce information traffic between Content Channels and Content Engines, Content Channels may access a content-based cache **168** in which information that was previously retrieved from Content Engines is cached. Engine communication link **170** is used by Content Access Modules to communicate with Content Engines. The protocol used is the native protocol of the Content Engine. For example, if the Content Engine is an SQL based database system then the protocol used may be a series of SQL commands. The Content Access Module is responsible for connecting the Content Engine to the QuestObjects System.

[0139] Content Engines **174** are the primary source of information in the system. Content Engines can be located on any physical computer system, may be replicated to allow load balancing, and may be, for example, a database, algorithm or search engine from a third-party vendor. An example of such an algorithm is Soundex developed by Knuth. Content Engines may require user authentication, which, if required, is handled by the Syndicator (through the Content Access Module).

[0140] The invention uses Content Engines as a source of strings. One skilled in the art would understand that a string may, for example, contain a URL of, or a reference to any resource, including images and movies stored on a network or local drive. Furthermore, strings may have metadata associated with them. In one embodiment, strings might have a language code, creation date, modification date, etc. An entry in a dictionary may have metadata that relates to its pronunciation, a list of meanings and possible uses, synonyms, references, etc. A thesaurus term may have a scope note, its notation, its source and its UDC coding as metadata, for example. Metadata of an encyclopedia entry may include its description, references, and links to multi-media objects such as images and movies. A product database may have a product code, category, description, price, and currency as metadata. A stock quote may have metadata such as a symbol, a company name, the time of the quote, etc. Instructions to a control system may contain parameters of those instructions as metadata. For example, the instruction to open a valve can have as metadata how far it is to be opened.

[0141] Further details of an embodiment of the system are provided below, and also in copending U.S. patent application Ser. No. 09/933,493, entitled "SYSTEM AND METHOD FOR ASYNCHRONOUS CLIENT SERVER SESSION COMMUNICATION", filed Aug. 20, 2001, and incorporated herein by reference.

QuestField Products

[0142] The QuestObjects technology was designed to be compatible with any platform, including traditional client/server environments, but it is especially powerful for applications developed for use in web browsers and on handheld wireless devices (cell phones, PDAs, etc.). Products based on the QuestObjects technology, called QuestFields, have significant technological features and competitive advantages, many of which have never before been available for web browser applications. These include:

[0143] Easy Integration—QuestFields are very easily added to existing HTML pages. Contrary to other rich Internet technologies, QuestFields can be implemented in most web browser applications without changing the existing application's source code or web page design. Moreover, using standard CSS (cascading style sheets), the borders and buttons of a QuestField can easily be modified to reflect each customer's individual style.

[0144] Discrete Components—A QuestField is comprised of standardized components that can easily be combined and reused. In addition, multiple dependent QuestFields deployed on a single web page can automatically share the same user session.

[0145] Ultra-Thin—QuestFields have been designed as high performance, ultra-thin clients that nevertheless offer the user extremely high functionality and friendliness. By keeping track of session information on the QuestObjects Server, a QuestField effectively acts as an efficient, continuously updated "window" on server data.

[0146] Field Dependencies—QuestFields can have dependencies on each other's data, enabling data in one field to be automatically updated after a change in another field. Moreover, this can be accomplished without any additional client-side or server-side coding. Dependencies can be created between any content channels. Thus, a QuestField querying an SQL database can be dependent on the values of a QuestField linked to an LDAP directory.

[0147] Ubiquitous—QuestFields can use Macromedia's Flash Player, which is now installed on 98% of the computers connected to the Internet. This means that more than 500 million computers can use any Flash-based QuestObjects product without installing any additional software. Nevertheless, a DHTML QuestField is currently under development to ensure that QuestFields continue to offer the highest performance for the most users.

[0148] Interchangeable—QuestFields can be implemented in any programming language. By simultaneously supporting multiple client technologies and by dynamically selecting the appropriate technology for a specific application user, QuestFields can be used by virtually all Internet and intranet users—a much wider user reach than any other Rich Internet Application technology.

Support for Web Services

[0149] Embodiments of the system may utilize web services to provide some or all functionality. Web services are open standards-based functional components that allow applications to connect over the Internet on demand, using loose coupling. Today, Web services are the core IT strategies of the computer industry's leaders, including IBM, Microsoft, and Sun. To address today's business needs, applications have an ever-increasing need to work closely together. By their very nature, Web services offer interoperability across all platforms that implement a Web services stack, regardless of programming language or operating system. Web services support this in a standard, well-defined manner. Web services created using Sun's J2EE-based technologies are fully interoperable with Microsoft's .NETweb services.

[0150] By definition, QuestObjects is a document-type Web service. MasterObjects has designed the product to be fully compliant to open standards.

QuestObjects System Implementation

[0151] As described herein, the QuestObjects implementation of the system is ideally suited for Intra-enterprise component reuse. QuestObjects enhances legacy applications, which often use proprietary connections to the database, by adding a second service-based mechanism to access the information. This can be done while maintaining current, proprietary invocation mechanisms. QuestObjects can also be used for Componentized E-Services (B2B), where information is provided through a QuestObjects Service on a subscription basis.

[0152] QuestObjects brings interaction to static Web pages. As a result, it enables Web applications to deliver to users, through a Web browser, much of the interactive richness of the user's typical personal computer or workstation. QuestObjects is a powerful yet simple concept that can be summed up in one sentence: "QuestObjects enables Web clients to interact with servers using string-based input on a per character basis."

[0153] FIG. 4 shows an illustration of an asynchronous session-based search system including a front-end client search field 190 and a back-end server datastore 192. QuestObjects' power lies in its ability to leverage the textual nature of the Web. Text is the basis of everything users find on the Internet. Consequently, Web applications are all about string-based data. The primary purpose of the Web is to find information, but whether the user is looking for a book to buy, searching train schedules, browsing holiday brochures, gathering data on company customers or employees, or getting stock quotes, all of the data presented to the user by the Web is delivered in the form of text—including images which are represented, in text, by their URL. Working together, it is rather astonishing what HTML and Web browsers can accomplish just with text, but the final result still falls far short of the user's typical personal computer and workstation experience because there is no interaction and no automatic, instantaneous feedback. Instead, new information is presented only after the user enters some string-based data into a form, presses the Submit button, and waits for the Web page to analyze the input and redraw the entire page.

[0154] By contrast, today's powerful, non Web-based applications that are accessed directly by personal computers and workstations provide a vast array of instant help and feedback. Automatic type-aheads, selection lists, wiggly lines indicating spelling mistakes and other interactive features provide users with real-time feedback based on real data in some kind of data store. Efficiently and effectively, QuestObjects adds this same kind of instant feedback to regular Web pages by linking user actions to data stored anywhere on the Internet.

[0155] Providing instant feedback is a powerful enhancement to Web applications, but the QuestObjects technology offers more. Multiple fields containing dependent data can automatically update themselves when one or the other changes. QuestObjects collects statistics, provides user histories, and allows accurate accounting of data returned to the user. QuestObjects does all this easily and transparently. Depending on the implementation, the product is no more than a black box between the user and content located anywhere on the Internet.

QuestObjects Architecture

[0156] QuestFields—QuestObjects enables dynamic interaction between client-side UI elements and back-end data. These enabled UI elements are called QuestFields. QuestFields are connected to a data source through a Content Channel.

[0157] FIG. 5 shows an illustration of an asynchronous session-based search system including multiple front-end client search fields 200, 202, multiple channels 204, 206, and a back-end server datastore 208.

[0158] Questlets—QuestFields live within a lightweight application called a Questlet. A typical Questlet is a little UI application, very much like a Java applet, an ActiveX component, or a Flash movie. Questlets are not intended to take over the HTML page. Rather, they enhance the typical behavior of related UI elements as might otherwise be used in prior art systems.

[0159] FIG. 6 shows an illustration of an asynchronous session-based search system including multiple front-end client search fields.

[0160] QuestObjects service providers bundle and deliver their services through Content Channels. Service providers are also known as Syndicators because they can subscribe to each other's Content Channels.

[0161] Questlets are separate from the Web application: They typically do not interact or interfere with current business logic. Questlets are connected to an existing Web application at only two points: the user interface and the data store.

[0162] As shown in **FIG. 6**, a Questlet **210** contains QuestFields **212**, **214** that replace some or all of the input fields in the Web application's user interface. QuestFields do not usually change the semantics of input values; they simply add dynamic interactivity to the application.

[0163] A Questlet and its QuestField(s) interact with the database to accommodate the interactive behavior of the user interface.

[0164] Because QuestObjects adds interactive behavior to client-side elements based on server side data, it might seem that network traffic would be substantially increased and performance degraded. However, network traffic is reduced to a minimum by sending partial datasets rather than complete ones. For example, suppose a Web application has an auto-completing QuestField using a Content Channel containing artist names in a music database. When a user starts typing, the QuestField requests data from the data source through the Artist Channel. If the first letter is a c, the result set containing all artists starting with a c is probably quite large. The QuestField can be configured to just ask, for instance, for just the first 20 artists. The QuestField retrieves more results as the user scrolls through the drop-down list contained in the QuestField.

[0165] Even if the QuestField asks for more results, only the difference between the QuestField's current data set and the QuestField's requested data set is transmitted over the network, thus further reducing traffic.

[0166] The QuestField's "data retrieval intelligence" has important architectural implications. The client is not the only location where the result set is maintained. The QuestObjects Server itself also maintains this information in order to calculate information to be sent to the client.

[0167] **FIG. 7** shows an illustration of an asynchronous session-based search system including a front-end client search fields **220**, a server **222**, result storage **224**, **226**, and a back-end server datastore **228**.

[0168] The QuestObjects system can be deployed over multiple tiers as shown in **FIG. 8**, which shows an illustration of a multi-tier asynchronous session-based search system including client tier **230**, server tier **240**, service tier **250**, and content tier **260**.

[0169] QuestObjects Client—The Client Tier runs the Questlet and its QuestField(s). It is typically deployed in a Web page, where a Web server such as Apache or Microsoft IIS serves the Questlet.

[0170] QuestObjects Server—The Server Tier manages client sessions. It maintains an administration for each QuestField. As mentioned above, this administration reduces network traffic.

[0171] The QuestObjects Server runs as a servlet in a standard servlet container such as Tomcat or JBoss.

[0172] QuestObjects Service—Service providers called Syndicators offer information disclosed by QuestObjects. The Service Tier runs these Syndicators, which provide their services through Content Channels. A Syndicator offers subscription-based access to its Content Channels for managed user groups. A Syndicator collects statistics and provides billing information.

[0173] Like the QuestObjects Server, QuestObjects Services run as a servlet in a standard servlet container.

[0174] Content Engine—Content is stored in another tier, called the Content Engine, which is usually located outside the QuestObjects system. Syndicators use Content Access Modules (CAMs) to link each Content Channel to a data store disclosed by the Content Engine. A CAM provides an abstraction layer between the QuestObjects system and any data store. QuestObjects currently includes CAMs that access any SQL-compliant database through JDBC or any enterprise directory through LDAP.

QuestObjects Clients

[0175] Client applications that take advantage of QuestObjects will usually comprise multiple windows or web pages. Each window or web page in a client application may use one or more QO Client elements: Visible or invisible client components that implement QuestObjects behavior. Some user interfaces may dynamically add or remove QO Client elements to a window (e.g., when a user opens a new tab pane in the application), at which time other QO Client elements may remain active, or are destroyed (e.g., when a web page is replaced by another page). QO Client elements may be only a small part of the client-side application, although some QO Clients may remain active during the entire life of the client application. A QO Client Controller may refuse a Quester (perhaps because it isn't connected to a server that has the Channel requested by the Quester) by passing it a reference to a different controller. This can be transparent to the QuestField.

[0176] Multiple QO Client elements in an application (or multiple client applications running in an OS or virtual machine) can share connections to the database. This is achieved by having them communicate through a single "QO Client Controller". The QO Client Controller is responsible for the QO Protocol (QOP), maintaining the Quester registry (including waiting requests and matching results), handling of dependencies (probably through a separate object), and request buffering for network optimization. When QO Client elements share a connection to the database, the connection is only established once. Additional QO Client elements an register themselves with an existing session; QO Client elements that are no longer used deregister themselves. The QO Client Controller is independent of QO Client elements: Conceptually, the Client elements find a suitable controller rather than the other way around. This ensures that a QO Controller may survive the life of QO Client elements: QO Client elements can be instantiated and destroyed during the life of a QO Controller. Client-server connection state (a.k.a. controller state) can be displayed by any QuestField that has a Quester connected to the corresponding QO Client Controller. In order not to unnecessarily complicate the user interface, only the first currently visible QuestField may actually display the connection state.

[0177] QO Client software is fault-tolerant. If communication errors occur, the client-side objects can function in off-line mode. An auto-completing QuestField, for example, will allow users to keep typing into the field, and the value in the entry field can still be submitted to the web application.

[0178] A QO Client's full state, including the current input buffer, results, selection state, etc, may be "frozen" and "restored" at a later date. This allows a QO Server to restore the QO Client after the client has been away. This could be necessary if the client application crashed, lost its connection, or if the client page was refreshed by using the back-button in a browser. A related feature is suspension and resumption of active sessions during client-side sleep.

[0179] A QO Client may allow users to store local preferences. For example, a user of an auto-completing QuestField may switch auto-completion on or off. Or a user of a drop-down list may set a preference to have the drop-down list open itself automatically as soon as results are received from the QO Server.

[0180] Through the concept of "String Types" and "Query Types", QO Clients may interrogate directory services for QuestObjects Services and Channels that return specific types of content using specific types of queries. This makes it possible to create clients that work with result sets that contain data in a known format, without making those clients dependent on a specific channel.

[0181] QO Servers and Content Services may send meaningful (textual) messages for display by the Client. The Client sends optional language information to the Server at registration, allowing Server or Service to return appropriate messages for the Client locale. Clients do not need to be aware of possible server-side messages and are thus generic.

[0182] A web-based QO Client is able to submit its input buffer (either the entered part or the auto-completed string), current result string, or selected result strings (or corresponding keys) to the server using the non-QO specific browser submit mechanism (through GET or POST), allowing users to keep using form fields in the "old" way. QO Clients can therefore be used to enhance existing web-based applications, without needing to rewrite server-side application code.

[0183] Depending on rules set by the Channel, a QO Client may validate a query before sending it to the server (note that the server must still know the latest input buffer in order not to send the previous query results to the client).

[0184] A QO Client may define a default (initial) value for its input buffer, causing the QO Server to perform a query as soon as the Quester is registered. This may be necessary to support browser "back" functionality, where a QuestField retrieves its previous input buffer value from a form input field.

[0185] A Client receives the expiration date/time for each result set, which may be overridden by individual strings in the result set. A client can use this to automatically re-query an expired result set or to automatically re-query metadata for strings that have expired (which may be necessary in a document management system that returns a list of documents, some of which may have very time-sensitive information).

[0186] A QO Client's value and query may depend on values in other fields in the application. If these other fields are also QO Clients, these dependencies may include: the key of a specific index in the result set; the string at a specific index in the result set; a set of keys for all selected indices in a result set; a set of strings for all selected indices in a result set; a set of keys in an index range; a set of strings in an index range; all result keys, or all result strings.

Web Page Integration

[0187] **FIG. 9** shows a method for using the present invention in systems that have limited technical capabilities on the Client side, such as, for example, web browsers with embedded Java applets, Flash movies, or other browser components or plug-ins. If developers of client systems have not integrated Client components of the present invention into their client software, then Client components needed for the present invention may be present as Plug-Ins, DLL's, or an equivalent device, or they can be downloaded to the client computer as applets. These applets can be written in the Java language, ActionScript, or other browser component language, when they are needed.

[0188] Although the system depicted in **FIG. 8** can be used to support clients in practically any server-based application server, and particularly in the case of a web server hosting an application used by end users to enter data that is partially retrieved using the present invention, the system is not limited to the web. The system provides an ideal solution for current web-based applications that consist of web browsers **300** on the client side and web host computers **302** with web server software **304** on the server side. To allow the web server to access data selected using the present invention, this system provides a link between the web server and the QuestObjects Server **306**. In this case, QuestObjects Server acts as a data-entry proxy between the existing client system (web browser) and the existing web server. Data entered by the client is submitted to the QuestObjects Adaptor instead of to the web server. The QuestObjects Adaptor then fills in the values of the Questers and passes the data to the web server. An Application Proxy is not required if the QuestObjects Client components can directly insert data into the client entry form on the web browser, as is the case on certain platforms that allow integration between Java applets or other components and JavaScript in the web browser.

[0189] In **FIG. 9**, the web server runs on a host computer **302**, typically associated with a fixed IP address or an Internet host name. The web server is accessed by any number of clients using web browsers **300**. To allow users to enter data and send data to the server, web pages make use of HTML forms **308**. To use the present invention, user interface elements such as entry fields in these HTML forms are associated with Questers **310** in the form of browser Plug-Ins, Java Applets, Flash Movies, or other browser components, or Client-script language implementations including QuestObjects Clients built in JavaScript or VBScript. Through a QuestObjects Controller **312** those Questers allow the user to access one or more QuestObjects Services hosted by a QuestObjects Server **306** using the protocol **314** of the present invention. The Server Controller **316** forwards user actions generated in the Client Questers **310** to their corresponding Server Questers **318** that thus are always aware of data selected in the Client. When a Server Quester is first activated, it checks whether it is being used by a client system that requires the use of an Application

Proxy. If the answer is yes, then the Quester creates 320 a corresponding AppHost Synchronizer 322 that contacts the QuestObjects Adaptor 326 on the host computer 302 using a standardized protocol 328. The QuestObjects Adaptor then knows which QuestObjects Server to contact to retrieve QuestObjects data 326 after the user submits form data 330 to the application host using the existing application protocol 332, such as HTTP POST or HTTP GET. The QuestObjects Adaptor then replaces the appropriate form field data with the strings selected in the Server Questers 318 before forwarding this form data, now including data selected using the present invention, to the web server 320, and thence to the client 322.

[0190] QuestFields may be easily and seamlessly integrated into a current Webpage. The first step is to determine which non-interactive HTML search fields are to be replaced by interactive QuestFields, as shown in **FIG. 10**. **FIG. 10** shows an illustration of a web interface 350 in accordance with the prior art.

[0191] In this example, the "Category"352 and "Search"354 fields are to be replaced with QuestFields, and because QuestFields are able to find and display records "on the fly", the "Search" button is now unnecessary and can be removed. To provide the user with additional information, a QuestField called "Album" is added.

[0192] The next step is to build the Questlet containing the QuestFields with the same look-and-feel as the target Web page. MasterObjects provides a default Questlet implementation built using Macromedia Flash. **FIG. 11** shows an illustration of a web-based search field 360 in accordance with an embodiment of the invention.

[0193] Finally, the client is added to the Web page after removing old form fields or after making them invisible. **FIG. 12** shows a listing of a html and JavaScript code 370 in accordance with an embodiment of the invention.

[0194] **FIG. 13** shows an illustration of a web-based search field 380 as it is used to receive data from a server in accordance with an embodiment of the invention.

Configuration

[0195] QuestObjects configuration is done using a straightforward text files for each Server, each Content Channel, and each database or directory connection. The configuration file of Content Channels that communicate through the JDBC Content Access Module includes the actual database queries with appropriate bind variables.

[0196] The configuration file of Content Channels that communicate through an LDAP connection contain the actual LDAP queries performed on the enterprise directory.

QuestObjects Protocol

[0197] QuestObjects uses a powerful protocol called the QuestObjects Protocol (QOP). QOP does not rely on the use of cookies and is designed to be compliant with existing Internet and Security standards. QOP is used for communication between Questlets and QuestObjects Servers. This is done transparently using XML in optional SOAP envelopes using HTTP(S) as the transport layer. Additional details defining an embodiment of QOP are provided below.

[0198] Security

[0199] QOP can be configured to run over SSL for complete security. Either the entire Web page, or just Questlet-Server communications can be securely encrypted. QuestObjects is designed so that neither users nor administrators need to worry about the details of the communication protocol.

Load Balancing

[0200] QuestObjects is designed for large Internet applications. The QuestObjects Server, QuestObjects Service and/or the Content Engine (database) can reside on a plurality of machines, allowing for load balancing and capacity expansion simply by adding more hardware. A QuestObjects Server uses "sticky" session connections. A client can logically connect to any server machine in the system. Once a session is established, all communications from the client IP address go to and from the same server.

Use of System for Interactive Database Searching

[0201] The system described above may be utilized in a Web, online, or similar environment, for purposes of interactive database searching, data entry, online purchasing, or other applications. This section describes how an embodiment of the invention may be incorporated into such an online environment.

[0202] **FIG. 14** shows a screenshot of a typical search screen interface 390 in accordance with the prior art, that may be used, for example, with a database application, an online application, or an online purchasing system. In this example, the interface is part of an online music store application, and allows a user to search for music records. As is typical with such applications, the user may select a category (in this instance from a pull-down list 392, although in other instances the user may select from a bullet-list). When the category has been defined, the user may enter their search criteria in the window 394 provided. While a category pull-down is not essential, it is commonly used in online and other environments to allow the user to narrow down the potential search results. When a category list is not provided, the system typically returns more hits than is desired.

[0203] **FIG. 15** shows a screenshot of the same search screen interface 390 in accordance with the prior art, illustrating the selection by the user of a particular category 396. One of the problems with the traditional interface is it provides no feedback to the user as to available options. Using the interface shown in **FIGS. 14 and 15**, all of the communication is one-way, i.e. from the user. But when the user selects a category, they have no knowledge as to whether there are any database records matching that category. Similarly, when the user enters a search criteria, there may be no matching hits. Furthermore, there is no feedback provided to the user that, for example, a more appropriate search might be useful, or that there may be slightly different spellings of that search term.

[0204] **FIG. 16** shows a screenshot of a search screen interface 400 in accordance with an embodiment of the present invention, illustrating how the QuestObjects technology can be used to assist a user with the search and selection of a database resource, and particularly address the feedback problems discussed above. Similarly, this example illustrates the interface as part of an online music store application, and allows a user to search for music records. In this embodiment a pair of pull-down lists are provided, one for Artist name **402** and one for CD name **404**. However, in other embodiments neither of the search fields may necessarily include a pull-down portion. Each search field indicates, in this instance by means of a small triangular arrow **406**, that the search field is enabled for use with the QuestObjects system. Depending on the embodiment, other indicators may be used, or indeed no indicator may be used.

[0205] **FIG. 17** shows a screenshot of a search screen interface in accordance with an embodiment of the present invention, illustrating how the system responds when a user enters data into a QuestObjects enabled search field. As shown in **FIG. 17**, as the user enters search data **410**, in this instance the first letter, or a few letters, of the Artists name, the search field displays an icon, in this instance a pair of rotating arrows, to indicate that the search field is communicating, via the QuestObject, search data to the server. The rotating arrow icon also indicates that the client is receiving corresponding information from the server as a result of the search data that has been sent.

[0206] **FIG. 18** illustrates the type of information that is dynamically returned to the user as they enter input data. Although there is no "submit" or similar button, since the client maintains a session with the server, and automatically sends and receives information from the server as data is entered, the server provides the client with increasingly appropriate information from the database. In the example shown in **FIG. 18**, as the user enters the text "r, o, . . . " etc. **414**, the server automatically responds with a list of records **416** matching this input data. In the embodiment shown, the records are presented as a list, from which the user may select one or more of those entries. Alternatively, if the desired record is not currently shown, the user can continue to enter input data to focus the search, and receive at the client more appropriate results.

[0207] **FIG. 19** shows the same example as the user enters more input data **418**. As the data is received, the server suggests increasingly more appropriate records **420** from which the user can select. In this manner the system may also be used to provide dynamically focused suggestions to the user.

[0208] **FIG. 20** shows the same example as the user enters more input data. **422** As the data is received, the server suggests increasingly more appropriate records from which the user can select. At this point the user has entered almost a complete Artist name. Again, as described above, the rotating arrow icon **412** indicates that input data is being automatically sent from the client to the server, while appropriate search records are retrieved for subsequent display on the client.

[0209] **FIG. 21** shows the same example as the user is presented with appropriate Artist name records **424** from the server, based upon the input data.

[0210] **FIG. 22** shows the same example as, this time the user has selected an Artist **426**, and is repeating a similar search sequence with the CD name **428**.

[0211] **FIG. 23** shows the same example as the user is presented with appropriate CD name records **430** from the server, based upon the input data.

Use of System for Interactive People Searching

[0212] **FIG. 24** shows a screenshot of a search screen interface **440** in accordance with an embodiment of the present invention, illustrating how the QuestObjects technology can be used to assist a user with the search and selection of a name database resource, for use in people searching. In one embodiment (shown as Option 1 in **FIG. 24**) a pull-down list **442** is provided for the persons Name. In another embodiment (shown as Option 2 in **FIG. 24**) a pair of pull-down lists **446**, **448** are provide for the persons Last Name, and First Name. As above, in other embodiments neither of the search fields may necessarily include a pull-down portion. Also as above, each search field indicates, by means of a small triangular arrow **450** or other device, that the search field is enabled for use with the QuestObjects system. Depending on the embodiment, other indicators may be used, or indeed no indicator may be used.

[0213] **FIG. 25** shows a screenshot of a search screen interface in accordance with an embodiment of the present invention, illustrating how the system responds when a user enters data into a QuestObjects enabled search field. As shown in **FIG. 25**, as the user enters search data **452**, in this instance the first letter, or a few letters, of the persons Name, the search field displays an icon, in this instance a pair of rotating arrows **454**, to indicate that the search field is communicating, via the QuestObject, search data to the server. The rotating arrow icon also indicates that the client is receiving corresponding information from the server as a result of the search data that has been sent.

[0214] **FIG. 26** illustrates the use of Option 1, and the type of information that is dynamically returned to the user as they enter input data. As above, although there is no "submit" or similar button, since the client maintains a session with the server, and automatically sends and receives information from the server as data is entered, the server provides the client with increasingly appropriate information from the database. In the example shown in **FIG. 26**, as the user enters the text "g, a, . . . " etc. **456**, the server automatically responds with a list **458** of name records matching this input data. In the embodiment shown, the records are presented as a list, from which the user may select one or more of those entries. Alternatively, if the desired record is not currently shown, the user can continue to enter input data to focus the search, and receive at the client more appropriate results.

[0215] **FIG. 27** illustrates the use of Option 2, and the type of information that is dynamically returned to the user as they enter input data. In the example shown in **FIG. 27**, as the user can enter either the Last Name **460** and/or the First Name **462** of the person. Matching records are returned using a similar process as described above.

Use of System for Other Applications

[0216] **FIG. 28** shows a screenshot of a complex search screen interface in accordance with an embodiment of the present invention, illustrating how the QuestObjects technology can be used to create a multi-level search interface, with multiple smart search fields or devices. In this example the search interface includes QuestObjects-enabled fields for First Name **482**, Last Name **484**, City **486** and Country **488**. As above, depending on the embodiment, the search fields may or may not include a pull-down portion. In the example shown in **FIG. 28** the pull-down lists also include pictorial representation of the field entry, making it more intuitive for the user. Also as above, each search field may indicate by means of a small triangular arrow or other device that the search field is enabled for use with the QuestObjects system. **FIG. 28** provides only an example of the type of interface that may be created using the QuestObjects system. It will be evident that a wide range of other interfaces may be similarly built with some or all of the QuestObjects features.

Variations on the Person Search Input Screen

[0217] **FIG. 29** shows a screenshot of an alternative person search input screen **490** in accordance with an embodiment of the invention. **FIG. 29** shows a screenshot of a formatted results list including email hyperlink buttons and an indication of the number of results found.

[0218] **FIG. 30** shows a screenshot of an alternative person search input screen **492** in accordance with an embodiment of the invention. **FIG. 30** shows a screenshot of an information pane that allows users to configure the QuestField and that shows QuestField- and Content-Channel-specific information and help.

[0219] **FIG. 31** shows a screenshot of an alternative person search input screen **494** in accordance with an embodiment of the invention. **FIG. 31** shows a screenshot of an About Box that shows technical information and copyright information for a QuestField.

Variations on the Types of QuestFields

[0220] Using the QuestObjects technology, at least six basic types of QuestFields can be created, some of which are shown in the above examples. These QuestField types differ in complexity, but they all have one thing in common: they can enhance any web browser or handheld wireless device application that is used to enter, find, retrieve and/or manipulate information stored in remote databases.

AutoLookup QuestField

[0221] **FIG. 32** shows a screenshot of an AutoLookup QuestField **502**. This is the simplest kind of QuestField. Upon user input (or after a dependent QuestField is modified), the QuestField does a "direct lookup" in the underlying content source where the data returned has a one-to-one relationship with the user input. Examples include a simple City QuestField that automatically displays the city for a specific Zip code, a Bank Number QuestField that verifies the validity of an account number, a Translation QuestField that automatically looks up the translation of text that the user has entered, a Stock Quote QuestField that returns a stock quote for a specific ticker symbol, or a Calculator QuestField that returns the result of a specific calculation performed on the user's input.

AutoComplete QuestField

[0222] **FIG. 33** shows a screenshot of an AutoComplete QuestField **504**. An AutoComplete QuestField assists the user during data entry by looking up multiple possible matches directly based on the user's character-by-character input. As the user types, the "best match" for the input is autocompleted into the input field. An optional popup list can display alternate choices to the user. The user input typically has a one-to-many relationship with the data that is returned by the content source, and the number of records returned is usually known. Examples include the PeopleFinder QuestField that looks up persons in a directory, a Product QuestField that helps the user find products, or an Account QuestField that helps the user in finding and entering customer account numbers.

AutoSearch QuestField

[0223] **FIG. 34** shows a screenshot of an AutoSearch QuestField **506**. An AutoSearch QuestField interprets the user input as a discrete search query that can be in any query format supported by the underlying search engine. The input is not usually autocompleted in the input field because of the nature of the input, although some AutoSearch QuestFields will suggest queries from a word-index or from a user query history list. Similar to the AutoComplete QuestField, search results are immediately displayed in a formatted popup list. The number of results returned from the server is typically unknown and limited by the search engine. Results in the AutoSearch QuestField popup list are usually filtered and ranked before they are displayed. Examples include a Site Search QuestField that enables users to find pages on a website based on full text Boolean searches, or a Document Search QuestField that allows users to retrieve documents or files based on full text as well as other criteria. A publishing company, for example, can use AutoSearch QuestFields to allow users to quickly and efficiently search newspaper and magazine archives.

Relational QuestField

[0224] **FIG. 35** shows a screenshot of a Relational QuestField **508**. A Relational QuestField provides a complex user interface consisting of multiple entry fields adapted for a specific use. A Relational QuestField simultaneously accesses multiple content channels and allows users to enter multiple values or click on results to "navigate" through relational content. Relational QuestFields provide a sophisticated user interface that typically feels like a "browser" or "navigator" because it can use multiple columns, tree lists, or even three-dimensional ways to display the results. Examples include an Address QuestField that can be used to enter full addresses (street, city, state, zip, etc), a Thesaurus QuestField that allows users to navigate through a taxonomy of terms, and a File Browser QuestField that behaves similar to Windows Explorer, yet operates efficiently and securely on remote content.

FreeForm QuestField

[0225] **FIG. 36** shows a screenshot of a FreeForm QuestField **510**. A FreeForm QuestField is a text area that allows users to enter blocks of text of any size. Rather than treating the entire input as a query, a FreeForm QuestField intelligently interprets the user input as it is typed, providing the user with wide range of "on the fly" text editing enhancements. Examples include a SpellCheck QuestField that checks and corrects the user's spelling or grammar based on remote dictionaries while the user is typing, or an AutoSave QuestField that automatically saves the user's input remotely while the user is typing into the browser.

Background QuestField

[0226] A Background QuestField does not have its own user interface. Instead, it is a QuestField that can be invoked to run in the background of an application, invisibly accessing a QuestObjects service. For example, a Background

QuestField could be a real-time stock price lookup function available to stored procedures in a relational database.

[0227] **FIG. 37** shows a table **512** that compares six basic QuestField types. From these basic types, complex QuestFields can be derived that combine the properties of multiple QuestField types.

Protocol (QOP) Implementation Details

[0228] This section describes in detail an embodiment of the QuestObject Protocol (QOP). As described above, the QuestObjects Client and the QuestObjects Server may communicate over the Internet using the QuestObjects Protocol. QOP is an application-layer protocol. Messages may be XML formatted. They can be transported in the body of HTTP(S) messages over TCP/IP, according to the HTTP specification. The implementation of QOP described here uses XML over HTTP, but other implementations of the protocol using different transport mechanism may be provided. To prepare for different physical variations of the protocol, the Adapter design pattern may be used in the software. The description below provides the minimum number of messages that are needed to implement QuestObjects functionality. The message names, element names, and attribute names provide a possible implementation, but other implementations of the protocol using different names are envisioned. To implement specific optional features of the QuestObjects technology such as pushing, additional messages may be implemented.

[0229] In a load balancing environment using the simplest implementation of QOP, the load balancer ensures that sessions are "sticky": QOP then assumes that communication from a specific QO Client takes place with a single QO Server instance. QOP may be mixed with other XML in the same HTTP request, or wrapped into a SOAP envelope (Simple Object Access Protocol). QOP XML can support full Unicode characters sets (typically using UTF-8 encoding). Each QOP XML message contains a block of QOP messages, for example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<messages xmlns="http://www.questobjects.com/qo/protocol/v10" ... >
  ...
</messages>
```

[0230] The messages block contains messages that validate against an XML schema. Elements in the messages block automatically adopt its default namespace. The recommended qop: namespace prefix is therefore optional, resulting in the smallest possible XML message size.

[0231] Each element of the QOP messages block is referred to as a QOP message. Every QOP message has a unique element name that logically describes its purpose. QOP messages and their attributes use so-called Wiki naming conventions: They start with a lowercase character and capitalize the first letter of each subsequent word. No consecutive capitals are ever used.

[0232] QOP messages sent from QO Clients are requests that have a unique messageIndex attribute containing a positive integer that is incremented for each message, starting at 1. The QO Server does not need to reorder requests that it receives according to the message index. It only uses the message index as an attribute in its reply so that the QO Client can match the reply to the original request. So, QOP relies on the fact that in most cases TCP/IP will deliver requests in their consecutive order. Since this is not guaranteed to happen, however, the QO Server may return an error or "no operation" results message to the QO Client if it receives a request that it cannot currently handle.

```
<anyRequestMessage messageIndex="12345" ... >
  ...
</anyRequestMessage>
```

[0233] Each QOP messages block that the QO Client sends to the QO Server (except the first messages block) includes a sessionId attribute containing the session id that was returned by the QO Server in the sessionStarted message.

```
<?xml version="1.0" encoding="UTF-8" ?>
<messages xmlns="http://www.questobjects.com/qo/protocol/v10"
        sessionId="4A74547885DAC49DDED925B6A0161BD5">
  ...
</messages>
```

[0234] Each QOP message sent from the QO Server is a reply to a QOP request from a QO Client. Each reply has a reply to attribute that matches the messageIndex of the corresponding request.

```
<anyReplyMessage replyTo="12345" ... >
  ...
</anyReplyMessage>
```

QOP Client Requests

startSession

[0235] This is a single QOP message sent by a QO Client to a QO Server in the very first request. It is used to create a unique session, which the QO Server confirms by returning a sessionStarted message. The QO Client must wait for sessionStarted until it can send any additional messages. If the QO Server cannot start a session, then it replies by returning an error message.

```
<startSession messageIndex="12345">
  <timeOffset>PT1078304323S</timeOffset>
  <clientControllerVersion>0.3.1</clientControllerVersion>
  <clientRuntimeName>Flash</clientRuntimeName>
  <clientRuntimeVersion>MAC 7,0,19,0</clientRuntimeVersion>
  <clientOs>Mac OS 10.3.2</clientOs>
  <clientBrowser>Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en)
  AppleWebKit/124 (KHTML, like Gecko) Safari/125</clientBrowser>
</startSession>
```

[0236] The value timeOffset is an XML duration indicating the QO Client's current time in seconds from 00:00 AM on Jan. 1, 1970. In the example implementation it uses standard XML notation for durations in seconds. clientControllerVersion is the QO Client communication controller's version number, formatted as a string consisting of major version (one digit), minor version (one digit) and patch version (one or more digits) separated by decimal points (periods). A QO Server can theoretically refuse to grant sessions to deprecated client controllers depending on this version string and any of the optional version elements (described below). The other startSession elements are optional. These elements have a default value and may thus be omitted. The QO Server uses the data for logging. clientRuntimeName is a string containing the name of the QO Client's runtime environments. In the example implementation, this is set to "Flash" and the default is an Empty string. clientRuntimeVersion is a string containing the version number of the QO Client's runtime environments. In the example implementation, this is the version string of the Flash Player and the default is an Empty string. clientOS is a string that contains the name and version of the QO Client's operating system with a default of Empty string. clientBrowser is a string that contains the name and version of the QO Client's web browser. It is only mandatory if the QO Client runs in a browser environment. Default value: Empty string.

registerQuester

[0237] This message is sent by the QO Client after it has received a sessionStarted reply. The QO Server replies by either returning a corresponding questerRegistered or an error message. Multiple registerQuester messages may be grouped into a single messages block. An optional block of requestDependency messages may follow these registerQuester messages, as long as all the registerQuester messages for Questers referred to in the dependencies precede the dependency request. The QO Server does not queue requestDependency messages: A requestDependency will fail if any of the corresponding registerQuester requests arrives later.

```
<registerQuester messageIndex="12345"
     name="com.masterobjects.artist">
  <channelId>artist</channelId>
</registerQuester>
```

[0238] The registerQuester name attribute contains the name for the Quester as it was configured in the Questlet. This name must be unique among Questers that share a connection (session) and should be unique among Questers in a client application. channel d contains a string that matches the name of the Content Channel configuration file on the QO Server.

requestDependency

[0239] This message is used by a QO Client to request a dependency on another Quester. The request must be sent after the registerQuester message for both Questers, but may be grouped into the same messages block. The reason this message works with Quester names instead of Quester ID's, is that this allows the dependency requests to be sent by the client before the corresponding questerRegistered have been received. The QO Server replies by either returning a corresponding dependencyGranted or an error message.

```
<requestDependency messageIndex="12345" name="artist-on-title">
  <questerName>artistQuester</questerName>
  <onQuesterName>titleQuester</onQuesterName>
  <trigger>inputBuffer</trigger>
</requestDependency>
```

[0240] The value of the requestDependency name attribute is a string that uniquely identifies the dependency. questerName is the name of the Quester that requests to be dependent on the Quester specified in on QuesterName. on QuesterName is the name of the Quester that should trigger the dependency. The requestDependency message must contain one or more trigger elements that define the events that trigger the dependency query. The element contains one of any number of constants reflecting dependency types.

Query

[0241] The QO Client sends query messages only after it has received the questerRegistered message for the corresponding Quester. The QO Server should either perform the query and return a results message containing data, refuse to perform the query and return a results message with a no-operation element explaining the reason, or return an error message if the query resulted in an error. Depending on user input speed, a Quester may send multiple query messages, but they should normally be sent as separate HTTP requests (thus containing a messages block with only a single query element), because the QO Server will only send a reply after processing all messages in the request.

```
<query messageIndex="12345" queryIndex="59">
  <questerId>5</questerId>
  <requestedRange from="0" to="2" />
  <string>be</string>
  <qualifier>CUSTOMER_APP_SESSIONID_ABCD OR
  WHATEVER</qualifier>
</query>
```

[0242] Each query sent by a Quester increments the queryIndex attribute, which is a positive integer, starting at 1 for the first query. The QO Server copies this number into the results message so that the client Quester can match those results to its original query. The QO Server will only return results for the latest query: it will return a results message containing a nop element if the query was skipped because a newer one was received.

[0243] The QO Server will return a results message containing a nop if the queryIndex received is lower than the previous queryIndex received for the same Quester. questerId is the id of the Quester as it was originally returned by the questerRegistered message. requestedRange is an empty element has two mandatory attributes from and to, which tell the QO Server which "view" of the result set the user wants. The first result in a result set corresponds to from value "0"; to is exclusive. The QO Server may return a results message containing fewer results than requested, and indicate that this matches the total number of available results. The QO Client displays the results and indicates to the user that no more results are available. The query process is finished. The QO Server may also return a results message containing the exact number of requested results or more: The QO Client displays them and the query process is finished. The QO Server may also return fewer results than indicated by the range, but indicate that the total number of results is bigger: The QO Client displays the received batch and may send a getRange message for the remainder of the result set. The string element contains the input string.

[0244] Qualifier contains the query qualifier, a string that can be used by a Content Channel to adjust its query. It is usually passed into the Questlet by the client application, or set by the user as a QuestField preference. Default value: Empty string.

getRange

[0245] A QO Client sends a getRange message to the QO Server to request a range of results. It is an optimized way for the Quester to ask for batches of results for the current query: The QO Server only returns results for results that were not yet sent in a previous results message for the same query (unless the server returns dropPrevious with value "1", meaning that previously sent results have expired). Otherwise, getRange results in the same behavior as a regular query message. A getRange should only be sent by the QO Client after receiving a reply to the corresponding query.

```
<getRange messageIndex="12345" queryIndex="60">
  <questerId>5</questerId>
  <requestedRange from="2" to="4" />
</getRange>
```

[0246] In each getRange message, queryIndex from the previous query or getRange is incremented by 1. The QO Server will only return results for the getRange with the highest queryIndex received from the QO Client; getRange requests with a lower queryIndex receive a nop reply.

stopSession

[0247] This message is sent by a QO Client to stop the session, thus allowing the QO Server to cleanup its corresponding resources. Note that, as always, the session id is derived from the sessionId attribute in the messages element. When the session is stopped by the QO Client, is includes the reason in the reason attribute, such as "submit", "disable", "suspend", "quit", or "unload" indicating that the Questlet stopped the session.

<stopSession message Index="12345" reason="submit"/>

QOP Server Replies

sessionStarted

[0248] The QO Server returns this message in reply to a QO Client's startSession message.

```
<sessionStarted replyTo="12345"
    id="4A74547885DAC49DDED925B6A0161BD5"
    urlSuffix=";jsessionid="
    timeout="PT3600S">
```

-continued

```
  <serverId>server1</serverId>
  <serverVersion>1.0.0</serverVersion>
  <serverBuild>RC1</serverBuild>
  <providerName>Reference   Customer</providerName>
  <providerDescription>Reference   Server</providerDescription>
</sessionStarted>
```

[0249] The id attribute contains the unique session id that was generated by the QO Server, identifying the unique client session. After receiving this id, the QO Client includes it as the value of sessionId in consecutive messages blocks that it sends to the QO Server.

[0250] The urlSuffix attribute tells the QO Client to append its value to the URL of any subsequent HTTP requests for the same session. timeout is an XML duration that informs the QO Client about the QO Server's session timeout. serverId is the unique id of the QO Server instance running on the server machine. serverVersion is the QO Server's version number, formatted as a string consisting of major version (one digit), minor version (one digit) and patch version (one or more digits) separated by decimal points (periods). A QO Client may theoretically refuse to work with a server that it knows does not support its features by checking this version string. serverBuild is a normalized string identifying the build of the server. For example "RC1" for release candidate 1. The other elements have a default value and may thus be omitted. providerName is a string that reflects the value that was configured on the QO Server. It will usually contain the name of the company that is hosting the server. providerDescription is a string that reflects the value that was configured on the QO Server. It will usually contain a textual description of the services provided.

questerRegistered

[0251] This message is sent as the reply to a successful registerQuester message. It tells the QO Client which id is to be used for any subsequent communication for the Quester, and passed textual information about the Content Channel back to the client.

```
<questerRegistered replyTo="12345" id="5">
    <channelName>Person Name Search</channelName>
    <channelHelpText>Enter the first characters of the first or last
    name of the person you are looking for.</channelHelpText>
    <channelCopyrightText>For use by Reference Customer
    employees only.</channelCopyrightText>
</questerRegistered>
```

[0252] The id attribute is a QO Server-generated id that is unique within the server session and replaces the Quester name as the unique identifier for the Quester used in subsequent client-server communication for the session. channelName is a human-readable string as it was defined in the Content Channel configuration. Along with providerName (which was returned by sessionStarted), this uniquely identifies the Content Channel to the user.

[0253] Optional questerRegistered elements channelHelpText and channelCopyrightText contain strings containing help text, copyright text and/or usage restrictions (as it relates to the content returned by the channel) to be displayed to the user.

dependencyGranted

[0254] This message is sent as a reply to the requestDependency message, confirming that the QO Server will include dependent data in any subsequent Content Query sent to the Content Access Module.

[0255] <dependencyGranted replyTo="12345"/>

results

[0256] The QO Server sends this message as the reply to a query or getRange request. The message either contains one or more ranges containing zero or more results, or a "no operation" element with an explanation of the reason why no results were returned. There is no guarantee that results messages arrive on the QO Client in the same order in which the QO Server sent them. Therefore, the QO Client must maintain a queue of results received, and handle them in the order of queryIndex, unless a new results message includes dropPrevious with value "1".

[0257] results message variant 1

```
<results replyTo="12345" queryIndex="1" dropPrevious="0">
        <range from="0" to="2" total="1000" isComplete=true
expires="PT107830438999S">
                <result>
                        <value>first result string</value>
                        <key>A57948</key>
                        <metadata>
                                <item>first metadata field</item>
                                <item>second metadata field</item>
                                <item>third metadata field</item>
                        </metadata>
                </result>
                <result> <value>second result string</value> </result>
        </range>
        <range from="3" to="5" total="1000" isComplete=true
expires="PT107830438999S">
                <result> <value>fourth result string</value> </result>
                <result> <value>fifth result string</value> </result>
        </range>
</results>
```

[0258] results message variant 2

```
<results replyTo="12345" queryIndex="1" dropPrevous="1">
        <nop reason="invalidQuery" />
</results>
```

[0259] The queryIndex attribute matches the queryIndex that was included in the corresponding query or getRange request. The optional dropPrevious attribute, which defaults to "0", indicates whether the QO Server wants the QO Client to forget any previous results that were returned for the same query. In that case, dropPrevious has value "1". The results message either contains one or more range blocks, or a single nop element. An empty nop element indicates to the QO Client that the QO Server did not perform the query, or that the server refused a getRange request. The mandatory reason attribute explains the reason why by including a constant value such as invalidQuery or querySkipped. Each range block contains zero or more result blocks and has three mandatory attributes and one optional attribute. The from attribute is the index of the first result of the range. The to attribute is the index one after the last result in the range (exclusive). The total attribute is the total number of available results. This is used by the QO Client to update the UI (usually, scroll bar) and possibly to automatically send another getRange message if fewer results are sent back in this range than requested while total indicates that more results exist. The optional is Complete attribute indicates whether the server retrieved the complete result set for the query in question or whether the result set was cut-off at a certain maximum size, in which case the actual number of results might be bigger than the amount specified in the total attribute. The optional expires attribute indicates the time at which the result expires, using the number of seconds since 1970 (corrected by the difference of timeOffset and QO Server time at receipt of the startSession message). If the value is "PT0S" or if the attribute is omitted, the results do not expire. For each result set entry that exists in the specified range, the range block includes a result element. If no results exist in the range, then the range block is empty. Each result contains a value element and optional key and metadata elements. The value element contains a string that was received from the Content Engine. The optional key element contains a string that uniquely identifies the result in the Content Channel. Using the value of key, elements value and optional metadata can be fetched from the Content Engine. The optional metadata block in each result element of a range contains an ordered set of item elements. Each item is a string that matches the corresponding metadata field as it was fetched from the Content Engine. As such, an item element may be empty to represent an empty metadata string, but an empty item is never omitted. The order of the item tags matches the order in which the Content Query was defined in the Content Channel definition. If no metadata is available for a result, the metadata block is omitted from the result element and the Quester considers all metadata items empty strings.

error

[0260] The QO Server returns an error message whenever it encounters a server-side error situation.

```
<error replyTo="12345" number="1001" sessionAlive="1">
        <errorText>An error has occurred in the QuestObjects
        server.</errorText>
</error>
```

[0261] Each error has a mandatory number attribute. The optional noSession attribute indicates that there is no (longer a) session after the error occurred (value "1"). The QO Client may reconnect by calling startSession again. If this attribute is omitted or "0", then the session is either still alive, or the QO Server does not know about session state. The latter is true if, for example, the server was unable to parse the request. The errorText element contains a human-readable string that is displayed to the user. It must include any leading and trailing punctuation marks.

sessionStopped

[0262] This message is sent to the QO Client after the QO Server has successfully closed its session and cleaned up the corresponding resources.

"""""<sessionStopped replyTo="12345"/>

Additional Optional Features—QuestObjects Client

Multi-Controller Dependencies

[0263] Multiple Client-side controllers (communicating to multiple Servers) are aware of each other so that they can register dependencies with each other. Multiple controllers on a Client communicate dependency triggers using a common notification center concept.

"Lossy" and "Non-Lossy" Pushing

[0264] Server denies a connection if it cannot guarantee queuing push replies between life beats, the interval of which is specified by the client, with a minimum enforced by the server.

Client Quester History

[0265] A Client Quester may have a history implemented as a cache in the Client controller.

Quester Context

[0266] A Quester has a unique name within its context (usually, window instance); dependencies assume the same context unless a specific context is specified.

Additional Optional Features—QuestObjects Protocol

Server Referral Mechanism

[0267] A "Dummy Server" can be installed on the "old" IP address/port so that Clients are automatically moved to a new Server. The Server sends a "Moved Permanently" message.

Client-Server Date Synchronization

[0268] Client and Server do not necessarily have to be set to the exact same date. Instead, the Client "tells" the Server its current date when it first connects, by sending a reference date, a time zone, and the number of seconds that have passed since that date. The Server will "translate" dates before sending them to the client, sending the number of seconds relative to the client's reference date.

Metadata Optional

[0269] The Client tells the Server whether metadata is required in each string transmitted to the Client. There are three moments at which metadata can be transferred to the Client: With every string in the result set, or With the current string in the result set (i.e., metadata is received automatically when a string is made the current string), or At request of the Client (the protocol allows the Client to request metadata for a range of strings). If metadata is not required, the Server only sends it to the Client at specific request.

"Metadata Displayed" Statistic

[0270] The Client tells the Server whether metadata for a string was displayed to the user or not.

Client-Side Caching

[0271] By calculating the difference between what's known on the Client and a new result set to be transmitted by the Server, Server-Client communication is limited. To this end, the protocol sends information to the Server about which result sets are still in memory on the Client. Whenever the Server (re-) sends a result set to the Client, it subtracts strings that are on the Client already. Client-Side "Keep Alive" Protocol

[0272] In order to allow the Server to send updates (updated result sets) to the Client, the Client sends a NOP package to the Server on a regular basis. This is different from the regular way in which results are received, where the Server tells the Client that a result set is not complete by sending a Server-side NOP and the Client simply waits for the remaining results.

Transmission of Dependency Data

[0273] A mechanism by which dependencies are sent to the Server and the mechanism by which the Server includes dependency data for transmission to the Service.

BLOB Communication

[0274] The QuestObjects protocol allows the Client to directly retrieve binary large objects from a Service. This is done through a separate Channel.

Welcome Messages

[0275] When registering, the Client tells the server whether it is capable of displaying Server or Service-generated messages. A "Welcome Message" sent to the Client by Server and/or Service has the "mustDisplayToUser" attribute. If this attribute is false, then the client may ignore the message.

Dialog Messages

[0276] Servers and Services may send meaningful (textual) messages for display by the Client. Client sends optional language information to the Server at registration, allowing Server or Service to return appropriate messages. Clients are not aware of possible server-side messages. Messages carry one of the following types: Information Only; Warning; Danger; Fatal Error.

License Agreement

[0277] A Service can force a client to ask the user to agree to a license agreement before a session becomes active.

Incremental Diffed Query String

[0278] A Client may perform a DIFF on its query string so it only sends incremental difference to the Server.

Conditional DIFFing

[0279] To reduce processor load, a diff is only performed over a certain package size.

Additional Optional Features—QuestObjects Server/Service

Automatic Server Discovery

[0280] A mechanism by which a client can discover Services using a DHCP-like mechanism such as UDDI or Rendezvous.

Password Security

[0281] Each Server has an optional name/password registry.

Inter-Server Dependencies

[0282] Dependency values are known on each Server and Server-side controllers exchange the actual dependency data, even for sessions on different servers.

Server Hopping

[0283] Two Servers can exchange their cache for a specific Channel, allowing them to synchronize their result set for a particular session that was moved from one Server to another.

Service Determines Cost of Query

[0284] A Service sends the cost of each query to the Server so it can be smart about caching the most expensive queries.

Client Session Influences Cache

[0285] The Server more likely caches a result set if any client session that used the result set is still alive.

Auto-Update Queries for "Pushing" Services

[0286] An auto-updating query is sent to the Service once until the reply was sent to the Client and the auto-update interval has passed.

Smart Auto-Complete on Server

[0287] If results are ordered in a known way, a Server may send a new result set to the Client if the query matches a previous "more global" query. In this case, the Service is not aware that a more specific query was performed.

Request Management

[0288] The QuestObjects server has a configurable request manager that manages the load of incoming client requests. By limiting the numberof request than can be executed in parallel, and queuing requests that could not be immediately performed, the request manager helps to ensure that the QuestObjects server remains responsive in high-load situations and that the server does not overload the content-source being queried with many simultaneous queries.

Unified Query Cache

[0289] In order to improve performance on recurring queries and to limit the load imposed on the content-sources being queried, the QuestObjects server caches results to user queries by storing them in cache that is common to all users. Each user has a view into the cache on the query that he/she performed. If more than one user performs the same query, there will be one result set stored in the cache and each of these users will only have a separate view on the query. Result sets are stored in the cache until they either expire, based on a result set expiration lifetime defined in each channel configuration file, or are evicted from the cache if it is full, using a least-recently-used cache algorithm.

Additional Uses and Applications

[0290] As a technology, QuestObjects™ has surprisingly broad potential. Applications include a wide range of markets including implementation on "connected devices" such as PDAs, set top boxes and cell phones. Other applications include those listed below. In applying the QuestObjects technology, existing applications may make use of enhanced functionality, including: Improved user friendliness; Improved speed of data entry; Improved quality of data entry; Quicker access to relevant data; Increased security of data entry; Increased security of data retrieved; Better control over usage patterns; Guarantee over strings displayed and used; and the ability Easily deliver and charge for content.

Auto-Complete, Type-Ahead:

[0291] In a typical application, users enter data into a field (entry field, cell phone, PDA) that automatically completes information typed. A client application can implement multiple QuestObjects™ components (called Questlets™ that possibly have multiple ways of querying the server using Quester™ instances, or QuestFields™ in specific situations) that share a connection to one or more QuestObjects™ servers. Multiple Questers™ can depend on each other for the data they retrieve. Depending on the client technology used, a Questlet™ can copy values to and from existing (web) forms or application windows.

Popup-Lists:

[0292] A QuestObjects™-enabled popup-list dynamically displays data that corresponds to a single query in the content engine. The associated Quester™ can take values from another QuestField™ so that the popup-list displays appropriate data that is continuously updated to reflect dependent data.

Look-Up, Finding Simple Reference Data (Dictionary, Thesaurus, Addresses, etc)

[0293] A (possibly invisible) QuestField™ returns a string with optional XML-formatted metadata. This XML-formatted metadata could be data displayed in a separate area of the Questlet™. A return string or its metadata could also contain a URL to data that is to be displayed. Depending on the client platform, data displayed could easily be in any multimedia format (image, sound, movie, streaming or not).

Usage Histories (URLs, Words Previously Used, etc.)

[0294] As an authenticated user uses QuestObjects™, the service tier stores usage histories. These are not only useful for doing statistics and invoicing, but can also be used for separate Questlets™ or Quester™ instances that disclose the usage history to the user or to an application that uses QuestObjects™ internally. In this scenario, a web browser could automatically store URLs visited in a QO service, so that the user has persistent access to the browser history, regardless of the workstation (or connected device) on which the browser is used.

Spell-Checking on the Fly

[0295] Since QuestObjects™ only transfers modifications to documents over the network, it can efficiently keep track of modifications made to text blocks. Dedicated spell-checking Questlets™ can be created that perform spell checking on-the-fly on any platform that supports QuestObjects™, including web browsers.

On-the-fly Verification of Credit Card, Bank Numbers etc.

[0296] Due to its inherent security features, QuestObjects™ technology can be used to check the validity of credit card (or similar) information while a user is entering data. The verification can take place before the entire web page is submitted to the server, or before database transactions are committed in a client/server system.

Decision Support Systems, Sensor Control Systems

[0297] QuestObjects™ services (content channels) continuously receive increasingly accurate information from client systems. This continuous nature (combined with the secure and fast communication protocol) allow decision support systems to immediately respond with appropriate actions to be taken by the client.

Verified and Guaranteed Display of Advertisements

[0298] Using pushing QuestObjects™ technology, Questlets™ provide a way to deliver content to client systems. Given certification of the client Questlet™, the content provider (service) knows exactly what part of the content was viewed and used.

Real-Time News Services

[0299] The pushing nature of QuestObjects™ allows real-time delivery of news to any client that supports the efficient QuestObjects™ protocol.

Composition of Ultra-Secure Documents (Continuously Backed up on the Server)

[0300] A QuestObjects™ service automatically receives each and every modification to client data. This provides a highly secure way of updating critical documents, where each modification is present on the server within a minimum amount of time. Persistent Questers™ can be restored automatically after recovery of a lost connection or after rebooting a crashed client.

[0301] The preceding detailed description illustrates software objects and methods of a system implementing the present invention. By providing a simple and standardized interface between Client components and any number of Content Engines that accept string-based queries, the present invention gives content publishers, web publishers and software developers an attractive way to offer unprecedented interactive, speedy, up-to-date and controlled access to content without the need to write an access mechanism for each content source.

[0302] In addition to acting as a standardized gateway to any content engine, the present invention can intelligently cache query results, distribute Services over a network of Servers, validate user and other client input, authorize user access and authenticate client software components as needed. These and other optional services are provided by the present invention without requiring additional work on the part of software developers or content publishers. Publishers can also keep track of usage statistics, on a per-user basis as required allowing flexible billing of content access. Content Access Modules allow software developers and vendors of Content Engines such as database vendors and search engine vendors to create simplified ways for developers and implementers of such content engines to disclose information through the present invention.

[0303] End users of the present invention experience an unprecedented level of user-friendliness accessing information that is guaranteed to be up-to-date while being efficiently cached for speedy access as the number of simultaneous users grows.

[0304] The present invention can be implemented on any client and server system using any combination of operating systems and programming languages that support asynchronous network connections and preferably but not necessarily preemptive multitasking and multithreading. The interface of the present invention as it appears to the outside world (i.e. programmers and developers who provide access to end users and programmers who provide Content Access Modules to Content Engines used by content publishers) is independent of both the operating systems and the programming languages used. Adapters can be built allowing the tiers of the system to cooperate even if they use a different operating system or a different programming language. The protocol of the present invention can be implemented on top of networking standards such as TCP/I P. It can also take advantage of inter-object communication standards such as CORBA and DCOM. The object model of the present invention can be mapped to most other programming languages, including Java, C++, C#, Objective C and Pascal.

[0305] Third-party vendors of software development and database management tools can create components that encapsulate the present invention so that users of those tools can access its functionality without any knowledge of the underlying protocols and server-side solutions. For example, a tool vendor can add an 'auto-complete field' to the toolbox of the development environment allowing developers to simply drop a Questlet into their application. In order to function correctly, the auto-complete field would only need a reference to the QuestObjects Server and one or more QuestObjects Services, but it would not require any additional programming.

[0306] The present invention may be conveniently implemented using a conventional general purpose or a specialized digital computer or microprocessor programmed according to the teachings of the present disclosure. Appropriate software coding can readily be prepared by skilled programmers based on the teachings of the present disclosure, as will be apparent to those skilled in the software art.

[0307] In some embodiments, the present invention includes a computer program product which is a storage medium (media) having instructions stored thereon/in which can be used to program a computer to perform any of the processes of the present invention. The storage medium can include, but is not limited to, any type of disk including floppy disks, optical discs, DVD, CD-ROMs, microdrive, and magneto-optical disks, ROMs, RAMs, EPROMs, EEPROMs, DRAMs, VRAMs, flash memory devices, magnetic or optical cards, nanosystems (including molecular memory ICs), or any type of media or device suitable for storing instructions and/or data.

[0308] The foregoing description of the present invention has been provided for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Many modifications and variations will be apparent to the practitioner skilled in the art. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, thereby enabling others skilled in the art to understand the invention for various embodiments and with various modifications that are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalence.

What is claimed is:

1. A system for session-based searching at a client for content at a server, comprising:

a communication protocol that provides an asynchronous session connection between a client and a server, and allows the client to send, as part of the same session, a plurality of consecutive query strings to query the server for content;

a client that transmits to the server within the session a plurality of queries to retrieve content from the server, wherein each of the plurality of queries are consecutive and form an increasingly focused query string for retrieving content from the server, and wherein each subsequent one of the plurality of queries extends the query string by one or more additional characters; and

a server that receives queries from the client, and in response to each increasingly focused query, automatically matches the increasingly focused query string against the content of the server, and returns increasingly relevant content to the client, for further use by the client within the same session.

2. The system of claim 1 further comprising a web browser including a web-based interface accessible at the client for creating queries, wherein the plurality of queries are entered into the web-based interface by a user to form an increasingly focused query string for retrieving content from the server.

3. The system of claim 1 wherein the client and the server communicate via the Internet using a hypertext transfer protocol.

4. The system of claim 1 wherein each of the plurality of queries are a single additional character to be added to the increasingly focused query string.

5. The system of claim 1 wherein each of the plurality of queries are a plurality of additional characters to be added to the increasingly focused query string.

6. The system of claim 1 further comprising a server repository for storing content information and for returning increasingly relevant content to the client in response to automatically matching the increasingly focused query string against the content of the server.

7. A method for session-based searching at a client system for content at a server system, comprising the steps of:

providing a communication protocol that provides an asynchronous session connection between a client and a server, and allows the client to send, as part of the same session, a plurality of consecutive query strings to query the server for content;

transmitting from the client to the server within the session a plurality of queries to retrieve content from the server, wherein each of the plurality of queries are consecutive and form an increasingly focused query string for retrieving content from the server, and wherein each subsequent one of the plurality of queries extends the query string by one or more additional characters; and

receiving at the server queries from the client, and in response to each increasingly focused query, automatically matching the increasingly focused query string against the content of the server, and returning increasingly relevant content to the client, for further use by the client within the same session.

8. The method of claim 7 further comprising receiving input at a web browser including a web-based interface accessible at the client, wherein the plurality of queries are entered into the web-based interface by a user to form an increasingly focused query string for retrieving content from the server.

9. The method of claim 7 wherein the client and the server communicate via the Internet using a hypertext transfer protocol.

10. The method of claim 7 wherein each of the plurality of queries are a single additional character to be added to the increasingly focused query string.

11. The method of claim 7 wherein each of the plurality of queries are a plurality of additional characters to be added to the increasingly focused query string.

12. The method of claim 7 further comprising providing a server repository for storing content information and for returning increasingly relevant content to the client in response to automatically matching the increasingly focused query string against the content of the server.

13. A system for asynchronous providing of information, comprising:

a server;

a database of content information coupled to said server;

a communication protocol that provides an asynchronous session connection between a client and the server, and allows the client to send, as part of the same session, a plurality of consecutive queries to query the server for content, wherein each of the plurality of queries form an increasingly focused query string for retrieving content from the database, and wherein each subsequent one of the plurality of queries extends the query string by one or more additional characters; and

wherein said server receives queries from the client, and in response to each increasingly focused query, automatically matches the increasingly focused query string against the database of content, and returns increasingly relevant content to the client, for further use by the client within the same session.

14. A people-searching system, comprising:

a server configured to receive requests from clients for people-searching content;

a database of biographic or other people-searching content information coupled to the server;

a communication protocol that provides an asynchronous session connection between a client and the server, and allows the client to send, as part of the same session, a plurality of queries to query the server for content, wherein each of the plurality of queries are consecutive and form an increasingly focused query string for retrieving content from the server, and wherein each subsequent one of the plurality of queries extends the query string by one or more additional characters; and

wherein the server receives queries from the client and in response to each increasingly focused query string, the query string being any of the first letters of a person last name, first name, or other personal information, automatically matches the query string against the database of content, and returns increasingly relevant content to the client, for further use by the client within the same session.

15. A system for searching for product-related information, comprising:

a server configured to receive requests from clients for product-related information;

a database of product-related information or other music content information coupled to said server;

a communication protocol that provides an asynchronous session connection between a client and the server, and allows the client to send, as part of the same session, a plurality of queries to query the server for content, wherein each of the plurality of queries are consecutive and form an increasingly focused query string for retrieving content from the server, and wherein each subsequent one of the plurality of queries extends the query string by one or more additional characters; and

wherein the server receives queries from the client and in response to each increasingly focused query string, automatically matches the query string against the database of product-related information or other content information, and returns increasingly relevant content to the client, for further use by the client within the same session.

16. The system of claim 15, wherein the product-related information is music content information and includes any of music title, artist name, and other music information.

17. A system for searching for documents in full-text databases, comprising:

a server configured to receive requests from clients for locations of documents including the full-text of those documents and metadata associated with those documents;

a database coupled to said server, that contains a full-text word index of said documents;

a communication protocol that provides an asynchronous session connection between a client and the server, and allows the client to send, as part of the same session, a plurality of queries to query the server for content, wherein each of the plurality of queries are consecutive and form an increasingly focused query string for retrieving content from the server, and wherein each subsequent one of the plurality of queries extends the query string by one or more additional characters; and

wherein said server is capable of receiving the increasingly focused query string from the client, said query string being any of the first letters of one or more indexed words with optional Boolean search operators, and as the query string is being extended, applying the query string against the database, and returning increasingly appropriate document locations to the client.

18. The system of claim 17 wherein each document is identified by a URL and wherein the full-text database contains a full-text index of a website on the Internet or on an intranet, and wherein the full-text index includes words found on web pages and in rich documents linked on the website.

19. A system for suggesting database records, comprising:

a server configured to receive requests from clients for content;

a database of content information coupled to said server;

a communication protocol that provides an asynchronous session connection between a client and the server, and allows the client to send, as part of the same session, a plurality of queries to query the server for content, wherein each of the plurality of queries are consecutive and form an increasingly focused query string for retrieving content from the server, and wherein each subsequent one of the plurality of queries extends the query string by one or more additional characters; and

wherein said server is capable of applying the increasingly focused query string against the database as it is begin extended, and suggesting increasingly appropriate content or search criteria to the client, for further use by the client within the same session.

20. A method of suggesting database records, comprising the steps of:

providing a database of content information coupled to a server;

accepting at the server requests from a client, via a communication protocol that provides an asynchronous session connection between a client and the server, and allows the client to send, as part of the same session, a plurality of queries to query the server for content, wherein each of the plurality of queries are consecutive and form an increasingly focused query string for retrieving content from the server, and wherein each subsequent one of the plurality of queries extends the query string by one or more additional characters; and

applying the increasingly focused query string against the database as it is begin extended, and suggesting increasingly appropriate content or search criteria to the client, for further use by the client within the same session.

* * * * *

EXHIBIT C

William Hassebrock

From:	William Hassebrock [whassebrock@venturecatalyst.net] on behalf of William Hassebrock [william.hassebrock@venturaverde.com]
Sent:	Thursday, September 04, 2008 11:33 PM
To:	'ericschmidt@google.com'
Subject:	Voice from the Past...
Categories:	!-Priority

Hi, Eric,

I hope you will recall our occasional interaction way back in the late 70s when I was President of the Princeton Club of Northern California and you were a grad student at Berkeley. Like you, I have spent my career trying to stay balanced on the bleeding edge of technology. I have been an entrepreneur, a venture capitalist, and now, as a "venture catalyst", I assist startups write business plans, find investors, and achieve traction.

My purpose in writing now is to let you know that for the past few years I have been working closely with a small software development firm that has invented an extraordinary new Internet search technology. I believe this technology could be of considerable interest to Google not only for your core search business but also for your new partnership with Verizon.

Briefly, this patent-pending technology, called QuestObjects, creates a powerful, ultra-thin client-server relationship between an individual field on a website, called a QuestField, and a remote server. One of our users referred to a QuestField as a universal Google Suggest on steroids.

A QuestField is a tightly integrated, end-to-end solution enabling any browser-based website or handheld device to perform incredibly fast search, retrieval and display of information from any remote database, including legacy databases. It is, in essence, a kind of miniature Citrix-like application that can be added to any website in a matter of hours with virtually no programming required – and its potential for handheld devices is perhaps even greater than its potential for websites.

The reason the world does not yet know about this technology, which I believe has the potential to revolutionize the user experience for both browsers and handheld devices, is because the company, MasterObjects, is quite small, is composed entirely of a few brilliant techies with little management or marketing experience, and is located in a small town in The Netherlands. Nevertheless, the company has scored a few major coups. For instance, Hewlett Packard, Siemens and several other corporations have been using the technology on their intranet sites for several years.

At this time, however, the only place the public can see the technology in action is on Princeton's website, a relationship that I helped to arrange. Although the implementation on the Princeton website is relatively simple, it will give you an idea of the power of the technology: http://www.princeton.edu/main/tools/az/a.xml?section=web.

Princeton is delighted with this enhancement to its website's search capability, but the QuestObjects technology is capable of much more as explained on the MasterObjects website: www.masterobjects.com.

MasterObjects has been working hard to create a stand-alone company for a number of years, but it has now decided that it needs a major partner/champion/godparent to help the technology get out of the lab and into the marketplace. For many reasons, most of them obvious, Google is at the top of our list of the small number of companies who might be possible suitors. Therefore, I thought I would alert you to MasterObjects and this extraordinary new technology.

If you at all intrigued, I would be pleased to arrange a teleconference for you or members of your team to speak with Mark Smit, the founder and CEO of MasterObjects. He can give you a remarkable demonstration of the power of the technology and tell you more about the company itself.

There is a good chance you already know or know of two other Silicon Valley pioneers who have been advising MasterObjects. Mark Medearis, of Heller Ehrman/Venture Law Group, is the company's corporate legal counsel. Marty Fliesler, of Fliesler Meyer, serves on the board and has been responsible for the company's IP strategy. You would certainly be welcome to speak with either Mark or Marty about the potential of MasterObjects.

Many thanks for allowing me to introduce you to MasterObjects, Eric. Please let me know if you would like to learn more.

Cordially,

-- Bill

William M. Hassebrock '68
President
Venture Catalyst Partners
616 Carolina Street
San Francisco, CA 94107

office: 1-415-821-7870
mobile: 1-415-235-3650
fax: 1-415-704-3375

Email: whassebrock@venturecatalyst.net