CONFIDENTIAL WORK PRODUCT SUBJECT TO NDA

To: **MasterObjects, Inc.**

From: **Spencer Hosie, Diane S. Rice**

Date: July 22, 2020

Re: Proposed Patent Infringement Lawsuit (Plaintiff: MasterObjects, Inc., Defendant: Amazon.com, Inc.)

This memorandum outlines the proposed patent infringement lawsuit to be filed by MasterObjects, Inc. against Amazon.com, Inc. in the United States District Court for the Southern District of New York. The lawsuit centers on the alleged infringement of U.S. Patent Nos. 8,539,024, 9,760,628, 10,311,073, and 10,394,866 (hereinafter referred to as the "Patents-in-Suit"). MasterObjects, Inc. contends that Amazon.com, Inc.'s predictive search technology, implemented in its Amazon.com website and mobile applications, unlawfully incorporates technology covered by the Patents-in-Suit, which has been confirmed through detailed investigation and analysis.

I. Background

Overview

MasterObjects, Inc. is a corporation organized under the laws of the State of Delaware, with its principal place of business in the Netherlands. The company is a pioneer in search technology, having developed a suite of innovative methods for asynchronous client-server session communication. These methods are protected by U.S. Patent Nos. 8,539,024, 9,760,628, 10,311,073, and 10,394,866, which collectively cover significant advancements in the efficiency and responsiveness of search systems.

IP Ownership

MasterObjects is the sole legal and rightful owner of the Patents. The patents cover various aspects of asynchronous communication between client and server systems, enabling real-time, incremental updates to search results as users type their queries. This technology significantly improves user experience by reducing latency and providing instantaneous search suggestions, which are crucial for modern search engines and e-commerce platforms.

IP Owner and Founder

MasterObjects was founded by Mark Smit, a visionary computer scientist who sought to overcome the limitations of traditional search technologies. In 1999 and 2000, Smit was frustrated with the slow and inefficient search processes of the time. He left his job and invested his life savings into founding MasterObjects, aiming to create a faster and more responsive search paradigm. Smit is the President and CEO of MasterObjects and his inventions, now protected by the Patents-in-Suit, have transformed the way users interact with search engines, enabling real-time results that improve both user satisfaction and system performance.

II. Patented Subject Matter and Infringement Allegations

U.S. Patent No. 8,539,024 pertains to groundbreaking methods in asynchronous client-server session communication. This patent outlines a system and method for managing real-time, bidirectional communication between a client and a server, enabling efficient and responsive data retrieval as a user types a query. The technology covered by this patent includes session-based communication, immediate synchronization of data entered on the client with the server, and sophisticated auto-completion functions. These innovations are crucial for applications requiring real-time data interaction, significantly enhancing user experience and system performance.

Upon conducting a meticulous examination, it has been determined that Amazon.com, Inc.'s predictive search technology, implemented on its Amazon.com website and mobile applications, infringes upon U.S. Patent No. 8,539,024. Amazon's technology unlawfully incorporates MasterObjects' patented methods, such as the session-based bidirectional communication and the auto-completion features that provide real-time search suggestions as users type. The detailed analysis reveals that Amazon's systems utilize the same real-time synchronization and data retrieval techniques outlined in the patent, thereby infringing on the patented claims.

U.S. Patent No. 10,311,073 further elaborates on the methods for asynchronous communication between a client and server. This patent introduces advanced features for handling real-time user inputs and delivering responsive suggestions and data updates. Key aspects of this invention include dynamic query handling and immediate data synchronization, which are essential for applications requiring instantaneous feedback and data processing.

Amazon.com, Inc.'s predictive search functionality also infringes upon U.S. Patent No. 10,311,073. The technology employed by Amazon involves dynamic query handling and real-time data updates, mirroring the patented techniques described in the patent. The infringement is evident as Amazon's system processes user inputs and provides immediate search suggestions, using methods that directly correspond to the patented claims. The inclusion of these infringing processes within Amazon's predictive search technology is well-documented in the attached infringement chart, highlighting the direct correspondence between the product's features and the patented claims.

In addition to the primary features, Amazon.com, Inc.'s predictive search technology incorporates several ancillary functionalities that further violate the Patents-in-Suit. These include advanced mechanisms for optimizing search query processing and delivering incremental search results, which are explicitly covered by the patents. Such techniques are integral to Amazon's competitive edge in the market by providing enhanced user experience through real-time search suggestions and data retrieval. The infringement analysis identifies these supplementary features as additional instances where Amazon.com, Inc. has unlawfully utilized MasterObjects' patented technology, underscoring the extensive nature of the infringement.

The evidence provided herein, supported by the detailed infringement chart, establishes a clear case for patent infringement by Amazon.com, Inc. The unauthorized use of the patented technology in Amazon's predictive search functionality not only violates the intellectual property rights of MasterObjects, Inc. but also undermines competitive fairness within the industry. Accordingly, it is recommended that legal action be initiated to address this infringement and seek appropriate remedies.

III. Prosecution and Validity

The prosecution history of U.S. Patent Nos. 8,539,024 and 10,311,073 reflects rigorous examination processes that ensured the issuance of robust and defensible claims. During the prosecution of U.S. Patent No. 8,539,024, MasterObjects disclosed and addressed prior art references, such as Kamvar, Baluja, and Kravets. These references were carefully reviewed by the patent examiner, who concluded that the claims of the '024 patent were novel and non-obvious despite the detailed scrutiny. The claims were meticulously drafted to encompass innovative methods of asynchronous client-server session communication, which were deemed patentable over the prior art.

The prosecution of U.S. Patent No. 10,311,073 was similarly thorough. During this process, references including U.S. Patent No. 5,222,234, U.S. Patent No. 5,737,734, and U.S. Patent No. 5,845,300 were considered. The examiner found that the claims of the '073 patent, which cover advanced methods for handling real-time user inputs and delivering responsive suggestions and data updates, were distinct and non-obvious in light of these prior art references. The prosecution history indicates that the claims were crafted to withstand invalidity challenges by clearly distinguishing the inventive features over the prior art.

Throughout the prosecution, the claims were refined and strengthened to ensure their robustness. For example, during the prosecution of the '073 patent, detailed arguments were presented to overcome objections based on prior art references. The patent owner argued that the combination of asynchronous communication with a cache system that stores query strings and search results

based on content queries from multiple users was not disclosed or suggested by the prior art. This argument was supported by detailed explanations and technical evidence, ultimately leading to the allowance of the claims.

The prosecution histories of these patents also involved the submission of continuation applications, which provide a strategic advantage by allowing the addition of new claims to further strengthen the patent portfolio. This proactive approach ensures that MasterObjects can adapt and expand its patent coverage in response to evolving technologies and market conditions.

The examination processes for these patents have resulted in the issuance of strong and enforceable claims. The plaintiff, MasterObjects, Inc., is well-prepared to address any questions or challenges related to the prior art. Detailed infringement charts and technical analyses are available to demonstrate how the patented technologies are distinct from the disclosed prior art and to reinforce the validity of the claims.

IV. MasterObjects Litigation Strategy

MasterObjects, Inc. intends to aggressively pursue its patent infringement claims against several prominent web and e-commerce technology companies, including Facebook Meta, Google, and Amazon. The attorneys have identified strong claims based on detailed analysis and previous infringement contentions. To maximize the impact and efficiency of these legal actions, MasterObjects plans to bring cases across three specific jurisdictions: the Northern District of California, the Eastern District of Texas, and the Southern District of New York. These venues are known for their expertise in handling complex patent litigation and have historically been favorable for patent holders.

The litigation strategy includes a carefully allocated initial budget of $2 million over the next year, with provisions for re-evaluation if any case proceeds to trial. This budget will cover filing fees, expert witness engagements, discovery processes, and other litigation-related expenses. The goal is to secure substantial settlements from Facebook Meta and Google, leveraging the strength of MasterObjects' patents and the clear evidence of infringement to negotiate eight-figure settlements. This approach allows for a cost-effective resolution while providing significant financial compensation for the unauthorized use of MasterObjects' technology.

In contrast, the attorneys are prepared to take the case against Amazon to trial, given the perceived high damages available. The detailed damages analysis indicates that the potential financial impact of Amazon's infringement could exceed $3.24 billion, making it a high-stakes case worth pursuing in court. MasterObjects' litigation strategy involves a thorough preparation for trial, including gathering robust evidence and expert testimony to substantiate the claims. The decision to potentially take Amazon to trial underscores the commitment to enforcing

MasterObjects' intellectual property rights and securing just compensation for the innovative technologies developed by the company.

V. Amazon's Likely Defense Strategies

Amazon is expected to challenge the validity of MasterObjects' patents on several grounds. However, MasterObjects is confident in the strength and robustness of its patent claims, fortified by a thorough prosecution history and careful claim drafting. Here, we anticipate and respond to Amazon's likely defense strategies based on their past contentions and the arguments raised in the PTAB proceedings and invalidity contentions.

PTAB Challenge and Outcome

Amazon, through its association with Unified Patents, previously challenged the validity of MasterObjects' U.S. Patent No. 10,311,073 in an inter partes review (IPR2020-01201). The PTAB denied the institution of the IPR, determining that the petition did not show a reasonable likelihood of prevailing on any of the claims. The PTAB concluded that the prior art references cited, including Kamvar, Baluja, and Kravets, did not sufficiently demonstrate that the claims were unpatentable. This decision reinforces the strength of the patent claims and suggests that similar arguments will be insufficient in future challenges [OBJ] [OBJ].

Anticipation by Prior Art

Challenge: Amazon is likely to argue that the patents-in-suit are anticipated by prior art references such as Kravets (U.S. 6,704,727) and Trower (U.S. 6,922,810). Amazon claims that these references disclose the predictive search functionalities described in MasterObjects' patents.

Response: MasterObjects' patents were meticulously examined during prosecution, with these specific references considered and distinguished. The PTAB's final written decision in the IPR proceedings rejected similar assertions, affirming the patentability of key claims over these references. This history demonstrates the novelty of MasterObjects' inventions and supports their ability to withstand anticipation challenges.

Obviousness

Challenge: Amazon may argue that the patents are obvious over combinations of prior art, including the patents cited in their invalidity contentions. They might assert that a person of ordinary skill in the art would have combined these teachings to arrive at the claimed inventions.

Response: The prosecution history reveals that the patent examiner considered the most relevant combinations of prior art and found the claims to be non-obvious. Moreover, the patents in question address specific technical challenges and provide innovative solutions not suggested or motivated by prior art, thereby reinforcing their non-obviousness. The PTAB's decision denying the IPR also supports the argument that the claimed inventions are not obvious.

Lack of Written Description and Enablement

Challenge: Amazon might assert that the patents fail to meet the written description and enablement requirements under 35 U.S.C. § 112. They may contend that the specifications do not adequately describe the claimed inventions or teach how to make and use them.

Response: The detailed specifications of the patents provide ample disclosure, describing the inventions in sufficient detail to enable a person skilled in the art to replicate them. The specifications include numerous examples and embodiments that illustrate the practical application of the claimed technologies. This comprehensive disclosure meets the requirements of 35 U.S.C. § 112, as confirmed by the successful prosecution history and the denial of the IPR.

Patent Eligibility under 35 U.S.C. § 101

Challenge: Amazon might challenge the patents under 35 U.S.C. § 101, arguing that the claims are directed to abstract ideas without an inventive concept. They may assert that the patents merely apply conventional search techniques in a client-server environment.

Response: The patents introduce novel methods for managing search queries and results that significantly improve the performance and user experience of search engines. These innovations go beyond mere abstract ideas, embodying practical applications that satisfy the requirements for patent eligibility under 35 U.S.C. § 101. The PTAB's decision to deny the IPR further underscores the substantive nature of the patented technologies.

MasterObjects is prepared to counter these challenges by leveraging the comprehensive prosecution history and robust claim construction that support the validity and enforceability of its patents. The anticipated defenses by Amazon have been addressed and refuted in various legal proceedings, and MasterObjects remains confident in its position.

VI. Damages

This discussion assumes that no pre-suit damages are claimed, although such damages may be available depending on the circumstances.

Amazon's predictive search technology, which is alleged to infringe upon the patents held by MasterObjects, Inc., holds a significant position in the market. Amazon's search functionality is integral to its e-commerce platform, which is accessible via the Amazon.com website and mobile applications. Given Amazon's dominant market presence, this technology plays a crucial role in enhancing user experience and driving sales across its vast product catalog.

Revenue Impact

Amazon's e-commerce platform generates substantial revenue, with billions of product searches conducted annually. The accused predictive search technology contributes significantly to this revenue by improving the efficiency and accuracy of search results, thereby increasing the likelihood of purchase. Assuming an average conversion rate of 3% for search queries and an average order value of $45, the financial impact of the infringing technology is considerable.

Sales Figures and Market Penetration

Amazon's e-commerce platform processes approximately 12 billion searches annually. With an estimated conversion rate of 3%, this results in 360 million purchases. Assuming the average order value is $45, the base revenue attributable to the predictive search technology is approximately $16.2 billion annually. Given that the patents-in-suit have been in force and allegedly infringed upon for at least five years, the cumulative revenue impact exceeds $81 billion.

Royalty Rate Calculation

To determine the potential damages, a reasonable royalty rate must be applied. Industry standards for licensing such technology typically range from 1% to 5% of the attributable revenue. Given the critical nature of the predictive search technology to Amazon's business model, a conservative royalty rate of 2.5% is reasonable. Applying this rate to the $81 billion in attributable revenue results in potential damages of approximately $2.025 billion.

Patent Expiration and Future Sales

The patents in suit are set to expire on various dates, with the latest expiring in 2031. Assuming the current rate of sales continues, Amazon is projected to generate an additional $48.6 billion in revenue from the infringing technology over the next three years. Applying the 2.5% royalty rate to this future revenue results in additional potential damages of $1.215 billion, bringing the total estimated damages to approximately $3.24 billion.

Expert Damages Analysis

An expert will be required to derive the appropriate royalty rate for the damages calculation. However, assuming a conservative royalty rate of 2.5%, the estimated lump sum damages would be approximately $3.24 billion. This figure will be adjusted to account for the net present value of future sales.

Historical Sales Data and Market Presence

Amazon's historical sales data and projected revenue for the next several years indicate a strong and growing market presence. Given Amazon's extensive user base and the integral role of its predictive search technology in driving e-commerce transactions, the financial impact of the infringing technology is significant. Dr. William R. Latham III, with his extensive background in economic impact analysis and expert testimony in patent infringement cases, will likely emphasize Amazon's market dominance and the substantial revenue generated by the accused technology.

Projection of Sales Growth

Dr. Latham's analysis will reassess the estimated lump sum damages favorably, highlighting the significant financial impact of Amazon's predictive search technology. He will likely point out that Amazon's projected revenue growth, driven by increasing e-commerce adoption and expanding product offerings, underscores the importance of the infringing technology. His expertise in econometrics and forecasting will be crucial in projecting future sales and adjusting the damages calculation to reflect the net present value of these future revenues.

Market Conditions and Competitive Advantage

Considering Amazon's sales growth and favorable market conditions, the estimated damages may be conservative. Dr. Latham will argue that the actual damages could be higher, reflecting the product's increasing market share and revenue potential. He will likely draw on his experience in analyzing the economic impacts of technology and innovation, emphasizing how Amazon's use of the infringing predictive search technology provides a competitive advantage that boosts its overall market performance.

Expert Testimony on Royalty Rate

Dr. Latham's extensive consulting experience with various industries, including technology and e-commerce, will bolster his testimony on the appropriate royalty rate. He will provide a detailed analysis comparing similar technologies and licensing agreements to justify the 2.5% royalty rate. His previous work in calculating economic damages in patent infringement cases will lend

credibility to his assessment, ensuring that the royalty rate is grounded in industry standards and reflective of the technology's value to Amazon's business.

Damages Conclusion

The evidence and calculations presented herein establish a strong case for substantial damages due to the infringement of MasterObjects' patents by Amazon's predictive search technology. The thorough analysis of Amazon's sales data, market penetration, and the critical role of the infringing technology in driving revenue support the calculation of significant damages. MasterObjects is well-positioned to seek appropriate legal remedies to address this infringement and secure compensation for the unauthorized use of its patented technology.

VII. Venue

The proposed lawsuit will be filed in the Southern District of New York (SDNY), which is a proper and favorable venue for several reasons. First, Amazon conducts substantial business within this jurisdiction, establishing sufficient ties to justify venue. Additionally, SDNY is known for its efficient handling of patent litigation cases, offering a streamlined process and knowledgeable judiciary that can expedite the resolution of the case.

It is anticipated that Amazon may attempt to transfer the venue to the Northern District of California (NDCal), potentially arguing convenience or jurisdictional challenges. To counter this, the plaintiff will emphasize the substantial connection Amazon has with SDNY, including significant sales and business operations within this jurisdiction. Furthermore, the plaintiff's choice of forum is generally given considerable weight, particularly when the forum has a legitimate interest in the matter.

MasterObjects intends to highlight several key points to maintain the chosen venue:

- Amazon's substantial revenue generated from sales in the SDNY region.
- The presence of major corporate offices and fulfillment centers within SDNY.
- Public statements by Amazon highlighting the importance of the New York market to their business operations.
- Previous litigation or legal engagements by Amazon within SDNY, demonstrating their familiarity and readiness to litigate in this jurisdiction.

MasterObjects recognizes that Amazon may argue for transfer to NDCal on the grounds that it is the location of their headquarters and many key witnesses. Amazon may also cite that NDCal is a hub for technology companies, potentially arguing that it has a more relevant and specialized legal environment for handling patent disputes involving complex technology.

Should Amazon succeed in transferring the case to the Northern District of California, MasterObjects intends to adapt by leveraging the following strategies:

- MasterObjects will emphasize the robust nature of its patents, which have already withstood scrutiny in previous legal proceedings, including those in NDCal.
- The plaintiff will prepare to engage local counsel experienced in patent litigation within NDCal to ensure effective representation.
- MasterObjects will also gather additional evidence and expert testimony that underscores the validity of its claims and the substantial impact of Amazon's alleged infringement, which remains unaffected by the change in venue.
- The plaintiff will seek to expedite the proceedings by leveraging the efficient case management systems in place within NDCal, aiming to minimize delays and move swiftly towards a resolution.

By preparing for both scenarios—maintaining the case in SDNY or adapting to NDCal—MasterObjects ensures a strategic and flexible approach to venue selection and litigation strategy. This preparation underscores MasterObjects' commitment to vigorously defending its intellectual property rights and securing appropriate remedies for the alleged infringement by Amazon.

Trials@uspto.gov
571-272-7822

Paper 26
Date: January 11, 2021

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

UNIFIED PATENTS, LLC,
Petitioner,

v.

MASTEROBJECTS, INC.,
Patent Owner.

IPR2020-01201
Patent 10,311,073 B2

Before KARL D. EASTHOM, ROBERT J. WEINSCHENK, and JON M. JURGOVAN, *Administrative Patent Judges*.

JURGOVAN, *Administrative Patent Judge*.

DECISION
Denying Institution of *Inter Partes* Review
35 U.S.C. § 314, 37 C.F.R. § 42.4

I. INTRODUCTION

A. Background and Summary

Petitioner, Unified Patents, LLC, filed a Petition requesting *inter partes* review of claims 1, 2 and 4–12 of U.S. Patent No. 10,311,073 B2 (Ex. 1001, the "'073 Patent"). Paper 1 ("Petition" or "Pet."). The Petition was accorded a filing date of June 30, 2020. Paper 3. Patent Owner, MasterObjects, Inc., filed a Preliminary Response ("Prelim. Resp.") on October 16, 2020. Paper 12. Petitioner filed a Reply on November 10, 2020. Paper 17 (public version). By Order dated December 1, 2020, we authorized additional discovery and briefing concerning the real parties in interest, and adjusted the briefing schedule accordingly. Paper 20. Pursuant to the Order, Patent Owner filed a Sur-Reply (Paper 22) on December 2, 2020, and Petitioner filed a Sur-Sur-Reply (Paper 25) (public version) on December 9, 2020.

Under 35 U.S.C. § 314(a), an *inter partes* review may not be instituted unless the information presented in a petition "shows that there is a reasonable likelihood that the petitioner would prevail with respect to at least 1 of the claims challenged in the petition." Upon consideration of the Petition and accompanying exhibits and evidence, we determine Petitioner has not established a reasonable likelihood that it would prevail with respect to at least one challenged claim in the *inter partes* review. Therefore, we deny institution of an *inter partes* review as to all of the challenged claims of the '073 Patent.

B. Real Parties in Interest

Petitioner identifies itself as the real party in interest. Pet. 86.

Patent Owner also identifies itself as the real party in interest. Patent Owner's Mandatory Notice, Paper 5, 2.

C. Related Matters

Patent Owner has asserted the '073 Patent in the following cases:

Case Name	Case Number	Court
MasterObjects, Inc. v. Facebook, Inc.	6:20-cv-00087-ADA	W.D. Tex.
MasterObjects, Inc. v. Amazon.com, Inc.	1:20-cv-3478-PKC	S.D.N.Y.

Patent Owner's Mandatory Notices, Paper 5, 2; Pet. 86-87.

The '073 Patent is related to the patents at issue in the following cases:

Case Name	Case Number	Court
MasterObjects, Inc. v Google, Inc.	4:11-cv-01054	N.D. Cal.
MasterObjects, Inc. v. Amazon.com, Inc.	3:11-cv-01055	N.D. Cal.
MasterObjects, Inc. v. Microsoft Corp.	3:11-cv-02402	N.D. Cal.
MasterObjects, Inc. v. Yahoo! Inc.	3:11-cv-02539	N.D. Cal.
MasterObjects, Inc. v. eBay, Inc.	3:12-cv-00680	N.D. Cal.
eBay Inc. v. MasterObjects, Inc.	18-2252	CAFC
MasterObjects, Inc. v Google, Inc.	4:13-cv-04304	N.D. Cal.
MasterObjects, Inc. v. Yahoo! Inc.	3:13-cv-04326	N.D. Cal.
MasterObjects, Inc. v Google, Inc.	14-1148	CAFC

MasterObjects, Inc. v Google, Inc.	4:15-cv-01775	N.D. Cal.
MasterObjects, Inc. v. eBay, Inc.	4:16-cv-06824	N.D. Cal.
eBay, Inc. v. MasterObjects, Inc.	IPR2017-00740	PTAB

Patent Owner's Mandatory Notice, 2; Pet. 86–87.

D. The '073 Patent

The '073 Patent, titled "System and Method for Asynchronous Retrieval of Information from a Server to a Client based on Incremental User Input," describes a system with a client part, a communication protocol, and a server part. Ex. 1001, code (54), code (57). The server part receives incremental text-based input from one or more objects in the client part, and asynchronously returns matching information to the client part from content sources such as databases or search engines. *Id.* at code (57).

Figures 13A to 13D of the '073 Patent, shown below, depict various states 230 of an input element on the client part.

FIG. 13A

FIG. 13B

FIG. 13C

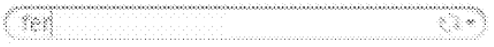
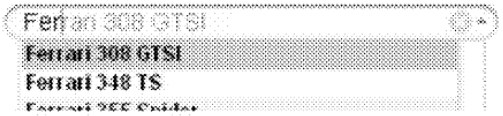


FIG. 13D



In Figure 13A, the user has not yet entered a query so the client part is not connected to the server. *Id.* at 21:61–64, 22:2–4. In Figure 13B, as the user enters characters of a query, the client part fires a query to the server. *Id.* at 22:2–12. In Figure 13C, the user has typed three characters ("Fer"). *Id.* at 22:21–24. In Figure 13D, the server responds with results corresponding to the input characters, which are displayed in a drop-down list.

Figure 12B of the '073 Patent, shown below, depicts a screenshot of a web page incorporating a client search object ("AutoComplete QuestField"). *Id.* at 3:24–26, 21:22–24.

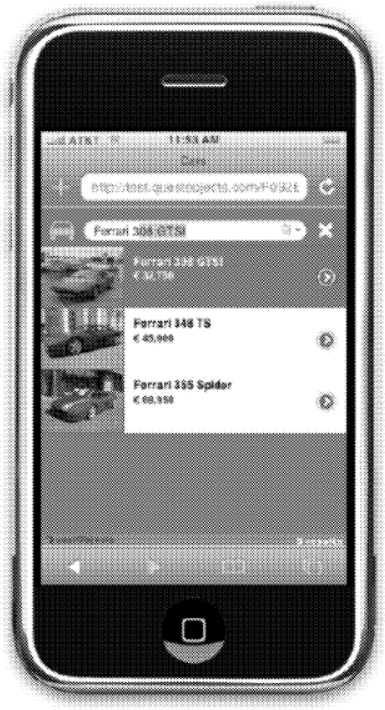


FIG. 12B

Figure 12B shows a screenshot of a device as it looks after the user enters "Ferrari 3." The screenshot lists various models corresponding to the entered input, and shows images of those models. Ex. 1001, 21:31–39.

E. Challenged Claims

Claim 1 of the '073 Patent is independent. Claims 2 and 4–12 depend, directly or indirectly, from claim 1. Claim 1 is set forth below with annotated numbering of elements:

> 1. [1preamble] A method, comprising:
>
> [1a] detecting, on a client computer, entry of a content search query into a field on a web page by a user;
>
> [1b] while the user is entering the content search query, automatically sending a string representing an incomplete search query to a server system comprising one or more computers;
>
> [1c] receiving, by the server system, the string;
>
> [1d] matching, by the server system, the string to entries in a cache of query strings and search results based on content queries received from multiple users, whereby cached search results contain a subset of data from one or more content sources;
>
> [1e] retrieving, by the server system, search result data for the incomplete search query;
>
> [1f] sending, by the server system to the client computer prior to completion of the search query at the client, a message containing information identifying the incomplete search query and at least a portion of the search result data that identifies the content in a content source;
>
> [1g] asynchronously receiving, on the client computer, without loading another web page and while the user is entering the content search query into the field, the message, and
>
> [1h] displaying at least a portion of the search result data on the client computer and enabling the user to retrieve additional content data corresponding to the search result.

Ex. 1001, 39:20–46.

F. Evidence

Petitioner relies upon the following prior art references (*see* Pet. 1), as well as the Declaration of Dr. Harley R. Myler (Ex. 1002)[1]:

Reference	Exhibit	Patent/Printed Publication
1005	Kamvar	U.S. Publication 2005/0283468 A1, published December 22, 2005
1006	Baluja	U.S. Publication 2006/0122976 A1, published June 8, 2006
1007	Kravets	U.S. Patent 6,704,727, issued March 9, 2004
1008	Porter	U.S. Publication 2007/0130131 A1, published June 7, 2007
1009	Barr	U.S. Patent 5,873,076, issued February 16, 1999

Patent Owner relies on the Declaration of Dr. Michael J. Pazzani (Ex. 2028).

G. The Asserted Challenges

Petitioner asserts the following grounds of unpatentability:

Ground #	Challenged Claims	35 U.S.C. §	References/Basis
1	1, 6–8 and 11	103	Kamvar
2	2, 4, 5, 9, 10 and 12	103	Kamvar, Baluja
3	1, 2, 4, 5 and 8–11	103	Kravets, Porter, Barr

Pet. 1.

[1] As discussed below, for purposes of this Decision, the claims at issue have an effective filing date prior to the effective date of the AIA's amendments to 35 U.S.C. § 112 (September 16, 2012) and 35 U.S.C. §§ 102 and 103 (March 16, 2013). Thus, we apply the pre-AIA versions of §§ 102, 103, and 112 in this Decision (pre-Leahy-Smith America Invents Act, Pub. L. No. 112-29, 125 Stat. 284 (2011) ("AIA")).

II. ANALYSIS OF CHALLENGES

A. Ground 1: Asserted Obviousness of Claims 1, 6–8 and 11 over Kamvar

1. Legal Background

For a patent to claim priority to the filing date of an earlier application, the specification of the earlier application must provide written description and enablement support for the patent's claims under 35 U.S.C. § 112 ¶ 1. That is, the earlier application must "contain a written description of the invention, and the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same." 35 U.S.C. §§ 112 ¶ 1, 120. A patent may claim priority through a chain of applications, but this requires continuity of written description and enablement support under 35 U.S.C. § 112 ¶ 1 in each of the applications back to the earliest asserted filing date. *Id.* This date may be referred to as the "priority date" or "effective filing date" of the patent.

Priority should be considered in context with the respective burdens of the parties. In an *inter partes* review, Petitioner bears the ultimate burden of persuasion regarding unpatentability, which never shifts to Patent Owner. *Dynamic Drinkware, LLC v. National Graphics, Inc.*, 800 F.3d 1375, 1379. Petitioner has the initial burden of production to show a reference is prior art. *Id.* The burden of production then shifts to Patent Owner to refute Petitioner's argument by either showing the prior art does not actually render the claims unpatentable or does not qualify as prior art (such as by showing that the patent at issue is entitled to priority to an earlier application that pre-dates a prior art reference). *Id.* The burden of production then shifts back to Petitioner to respond to Patent Owner's argument. *Id.* The Board then

evaluates all of the evidence and determines whether Petitioner has satisfied its burden of persuasion regarding unpatentability. *Id.*

At the institution stage, the Board has applied a similar burden-shifting approach with respect to the effective filing date of the challenged claims. The framework we apply here was explained in *Polaris Wireless, Inc. v. TruePosition, Inc.*, IPR2013-00323, Paper 9 (PTAB Nov. 15, 2013):

> In an *inter partes* review, the burden is on Petitioner to show a reasonable likelihood that it would prevail on a ground of unpatentability. With respect to entitlement to earlier effective filing dates, the Patent Owner is not presumed to be entitled to the earlier filing dates of ancestral applications which do not share the same disclosure. But, the issue first has to be raised by Petitioner in its petition, by identifying, specifically, the features, claims, and ancestral applications allegedly lacking § 112, first paragraph, written description and enabling disclosure support for the claims based on the identified features. Then, the Patent Owner has to make a sufficient showing of entitlement to earlier filing date or dates, in a manner that is commensurate in scope with the specific points and contentions raised by Petitioner.

Id. at 29; *see also Huawei Techs. Co. v. Samsung Elecs. Co.*, IPR2017-01980, Paper 9 at 9–10 (PTAB Feb. 27, 2018) (discussing *Dynamic*); *Franklin Elec. Co. v. Liberty Pumps, Inc.*, IPR2017-00113, Paper 14 at 12–13 (PTAB Apr. 27, 2017) (same); *Lupin Ltd. v. Pozen Inc.*, IPR2015-01775, Paper 15 at 10–11 (PTAB Mar. 1, 2016) (same).

2. Effective Filing Date of the '073 Patent

The '073 patent's filing date is February 17, 2017. Through a granted petition to correct its claim of priority (*see* Ex. 1004, part 1 of 3, 1–25), the '073 patent asserts priority benefits under 35 U.S.C. § 120 to earlier filing dates as a continuation of U.S. Patent Application No. 12/176,984, filed July 21, 2008 ("'984 Application"), which is a continuation-in-part of

Application No. 11/257,912, filed October 25, 2005 (Ex. 1011, the "'912 Application"), which is a continuation-in-part of Application No. 09/933,493, filed August 20, 2001 (Ex. 1010, "'493 Application").[2]

As mentioned, for purposes of institution, Petitioner has the burden of showing sufficiently that the patents or publications relied upon are prior art. *See Dynamic, supra.* In ground 1, Petitioner alleges "*Kamvar* was filed on June 22, 2004, and published December 22, 2005, making it prior art to the '073 patent under at least §102(e)." Pet. 5; Ex. 1005. Specifically, Petitioner contends that "[t]he claims of the '073 Patent are not entitled to the . . . priority date of the 2001 ['493] application" because it "does not provide support for '*a cache of query strings and search results based on content queries received from multiple users*,' as recited by claim 1 of the '073 Patent, on which all other claims depend."[3] Pet. 3–5. Petitioner alleges that the claimed feature was added as new matter in the later-filed '912 Application and '984 Application. *Id.* at 4.

To the contrary, Patent Owner contends that the "*cache*" limitation is supported by the '493 Application. Prelim. Resp. 10–22. Patent Owner notes that, "[t]o claim priority to a prior application, that application must 'describe an invention, and do so in sufficient detail that one skilled in the art can clearly conclude that the inventor invented the claimed invention as

---

[2] Although the '493 application claims priority under 35 U.S.C. § 119(e) to provisional application 60/951,850 filed July 25, 2007, Patent Owner does not assert priority to the provisional filing date in this proceeding. *See generally* Prelim. Resp.

[3] For the sake of brevity, this claim element is referenced as the "*cache*" limitation at points in the remainder of this decision.

of the filing date sought.'" *Id.* at 14 (citing *Lockwood v. Am. Airlines, Inc.*, 107 F.3d 1565, 1571–1572 (Fed. Cir. 1997)). "[T]he exact terms need not be used in *haec verba*," rather "the specification must contain an equivalent description of the claimed subject matter." *Id.*

We find Patent Owner's evidence shows sufficiently that the claim limitation of "*a cache of query strings and search results based on content queries received from multiple users*" is supported by the '493 Application. As Patent Owner notes, § 112 ¶ 1 support does not require literal support, word for word, in the priority application, but only that the equivalent description is present there. *See Lockwood, supra*.

Patent Owner notes that the "Content-based Cache" 222 (see Figure 2 of the '493 Application) is defined as "[a] persistent store of Queries and corresponding Result Sets executed by a Content Engine for a specific Content Channel." Prelim. Resp. 14–15 (citing Ex. 1010, 18, 27; Ex. 2028 ¶ 38). This establishes that the '493 Application provides § 112 ¶ 1 support for a server-side *cache of content queries and corresponding search results*.

Patent Owner further notes that the '493 Application states that "[e]nd 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*." Prelim. Resp. 15 (citing Ex. 1010, 55; Ex. 2028 ¶ 39) (emphasis added). This statement indicates that the cached queries are received from *multiple users*. Patent Owner's other evidence also supports this conclusion. *See* Prelim. Resp. 18 (citing Ex. 1010, 34; Ex. 2028 ¶¶ 41–42), 19–20 (Ex. 1010, 2–4, 13–14; Ex. 2028 ¶¶ 44–45).

Patent Owner reproduces Figure 8A of the '493 Application, which depicts objects "QoResultsCache" and "QoResultsCacheEntry" in cache embodiments that include query strings and search result sets received from multiple users. Prelim. Resp. 16-17 (citing Ex. 1010, Fig. 8, ¶ 117; Ex. 2028 ¶¶ 46–47). Patent Owner thus demonstrates sufficiently that the cache includes *query strings* and their corresponding search results from queries of multiple users.

"When neither the PTO nor the Board has previously considered priority, there is simply no reason to presume that claims in a CIP application are entitled to the effective filing date of an earlier filed application." *PowerOasis, Inc. v. T-Mobile USA, Inc.*, 522 F.3d 1299, 1305 (Fed. Cir. 2008). Here, Patent Owner has demonstrates that the '493 Application supports the "*cache*" limitation under § 112 ¶ 1. Prelim. Resp. 21–22 (citing Ex. 2028 ¶¶ 50–54). Further, Patent Owner's expert testifies that support for the "*cache*" limitation is present in each application from the '493 Application to the '073 Patent. Prelim. Resp. 21–22 (citing Ex. 2028 ¶¶ 50–54). In addition, each application in the chain of priority applications incorporates the prior applications by reference, so the '073 Patent cumulates the written description and enablement disclosures of the earlier applications. Ex. 1001, code (63), 1:7–30; Ex. 1004, part 1 of 3, 125 (the '984 Application), 167, 198 ¶ 1, 201 ¶ 1 (the '912 Application), part 2 of 3, 135–136 ¶ 2 (the '073 Patent).

For the foregoing reasons, Patent Owner's evidence shows sufficiently that the '493 Application provides support under § 112 ¶ 1 for the claim limitation of "*a cache of query strings and search results based on content queries received from multiple users*" as recited in claim 1, and

thereby all dependent claims, of the '073 Patent. As Petitioner's only priority contention alleged with specificity is that the "*cache*" limitation is missing from the '493 Application, and Patent Owner has shown sufficiently that the equivalent disclosure is present there, Patent Owner has shown sufficiently that the '073 Patent is entitled to a priority date of August 20, 2001, for purposes of this Decision.

3. Conclusion for Ground 1

In ground 1, Petitioner asserts Kamvar as prior art against claims 1, 6–8, and 11 of the '073 Patent. Pet. 1, 5–38. Petitioner asserts that Kamvar, filed June 22, 2004 and published December 22, 2005, is prior art under 35 U.S.C. § 102(e). Pet. 5; Ex. 1005, code (22), code (43). Petitioner argues that the '493 application, to which the '073 Patent claims priority, fails to support claim 1's limitation of "*a cache of query strings and search results based on content queries received from multiple users*" under § 112 ¶ 1. Patent Owner, however, sufficiently demonstrates that the '073 Patent supports the "*cache*" limitation of claim 1 under § 112 ¶ 1. Therefore, in a manner that is commensurate in scope with the specific points and contentions raised by Petitioner, Patent Owner sufficiently shows that the '073 Patent is entitled to priority of the '493 Application's filing date of August 20, 2001, which pre-dates Kamvar's filing date of June 22, 2004.

Accordingly, Petitioner fails to meet its burden to show that Kamvar is prior art to the '073 Patent. For this reason, ground 1 of the Petition fails to present a reasonable likelihood of prevailing in showing that at least one claim of the '073 Patent is unpatentable.

B. Ground 2: Asserted Obviousness of Claims 2, 4, 5, 9, 10 and 12 over Kamvar and Baluja

In ground 2, Petitioner asserts Kamvar in combination with Baluja as prior art against claims 2, 4, 5, 9, 10 and 12 of the '073 Patent. Pet. 1, 38–55. As noted above in Section II.A, Petitioner argues that the "*cache*" limitation of claim 1 of the '073 Patent lacks support under § 112 ¶ 1 in the '493 Application. Patent Owner, though, has shown sufficiently that the "*cache*" limitation of claim 1 of the '073 Patent has written description and enablement support under § 112 ¶ 1 and thus, for purposes of this Decision, is entitled to the priority date of August 20, 2001 established by the '493 Application's filing. Because Kamvar's filing date is June 22, 2004 and thus after the '073 Patent's priority date of August 20, 2001, Petitioner fails to demonstrate sufficiently that Kamvar is prior art to the '073 Patent.

Petitioner asserts that Baluja, filed December 3, 2004 and published June 8, 2006, is prior art to the '073 Patent under 35 U.S.C. § 102(e). Pet. 38; Ex. 1006, code (22), code (43). As discussed above, for purposes of this Decision, the '073 Patent's priority date is August 20, 2001, which is before Baluja's filing date of December 3, 2004. Therefore, Petitioner does not meet its burden to show that Baluja is prior art to the '073 Patent. Prelim. Resp. 30–31; Ex. 1006.

Accordingly, ground 2 of the Petition fails to present a reasonable likelihood that at least one claim of the '073 Patent is unpatentable.

C. Ground 3: Asserted Obviousness of Claims 1, 2, 4, 5 and 8–11 over Kravets, Porter and Barr

In ground 3, Petitioner asserts the combination of Kravets, Porter and Barr against claims 1, 2, 4, 5 and 8–11 of the '073 Patent. Pet. 1, 56–86. Petitioner asserts that Porter, a published application, claims priority to a

provisional application filed on November 21, 2000, and is prior art to the '073 Patent under § 102(e). Pet. 56; Ex. 1008, code (22), code (43). Petitioner contends Dr. Myler's testimony shows § 112 ¶ 1 support in the Porter provisional application (Ex. 1013) for Porter's claim 7 in asserting that Porter is prior art to the '073 Patent as of the Porter provisional application's filing date. Pet. 56–57 (citing Ex. 1002 ¶¶ 82–83).

Patent Owner contends that the Petition must show (1) that Porter's provisional application provides § 112 ¶ 1 support for the claims of Porter (the published application); and (2) that Porter's provisional application supports the allegations in the Petition. Prelim. Resp. 34–35 (citing *Dynamic*, 800 F.3d at 1380, 1382; *Intex Recreational Corp v. Team Worldwide Corp.*, IPR2018-00859, Paper 128 at 26 (PTAB Oct. 21, 2019); *In re Giacomini*, 612 F.3d 1380, 1383 (Fed. Cir. 2010)).

Patent Owner further contends

> Petitioner cites to Porter as teaching at least one limitation of every challenged claim in Ground 3. However, there is not *any* overlap between the cited portions of Porter and the provisional. Throughout Ground 3, Petitioner cites Porter Figs. 2, 4 and ¶¶5, 15, 19, 20, 23, 24, 27, 30, 32, 33, 39, 41. None of those figures or paragraphs are in the provisional. *Compare* EX1008 *with* EX1013. At no point does Petitioner attempt to identify an equivalent disclosure in the provisional. Thus, for this additional reason, Petitioner has not shown that any of the Porter disclosure it cites is entitled to the priority date of the provisional. This is fatal to Ground 3.

Id. at 38.

Although Petitioner attempts to show that Porter's provisional application supports claim 7 (Ex. 1002 ¶¶ 82–83), we agree with Patent Owner that Petitioner fails to show that the provisional application supports the portions of Porter relied upon to challenge the '073 Patent's claims. *See*

Giacomini, 612 F.3d 1380, 1383 (Fed. Cir. 2010) ("an applicant is not entitled to a patent if another's patent discloses the same invention, which was carried forward from an earlier U.S. provisional application or U.S. non-provisional application"). As Patent Owner notes above, there are significant differences between Porter and its provisional application, and one cannot assume that disclosure in Porter is necessarily present in the provisional. *Compare* Ex. 1008 and Ex. 1013. Therefore, Petitioner fails to meet its burden to show that Porter's provisional application supports the portions of Porter's published application relied upon in ground 3. *See Dynamic*, 800 F.3d at 1380, 1382.

In ground 3, Petitioner relies on Porter to disclose search-engine data and displayable data, which allows for displaying search results immediately without requiring selection of a link, for limitations [1d] to [1h] of the '073 Patent. Pet. 64–80. Without Porter, Petitioner's mapping fails to show that each element of the claims is disclosed in the prior art because Kravets's HTML/JavaScripts are not the displayable search result data recited in limitations [1d] to [1h] of the '073 Patent. Pet. 64. Petitioner relies on Barr merely to disclose a "query identification number" to identify a search query of a user. Pet. 77. The Petition does not show sufficiently that the combination of Kravets and Barr, without Porter, renders obvious all the limitations of claim 1. In addition, Petitioner relies on Porter alone, or with Kravets, to teach the limitations of claims 2, 4, 5 and 8–11 of the '073 Patent. *See* Pet. 80–86. Without Porter, Petitioner's mappings fail to show sufficiently the obviousness of these dependent claims.

Accordingly, Petitioner fails to show that Porter is prior art to the '073 Patent, and ground 3 of the Petition fails to present a reasonable likelihood that at least one claim of the '073 Patent is unpatentable.

III. REMAINING MATTERS

Because we have decided to deny the Petition for the reasons discussed above in Section II, we do not reach the remaining issues raised in the briefings. These matters include arguments concerning motivation to combine the alleged prior art references (Prelim. Resp. 42–43, 48); claim elements alleged to be missing from the asserted prior art (Prelim. Resp. 23–30, 31–34, 39–49); alleged lack of particularity in the challenges (Prelim. Resp. 29, 31, 33–34, 39, 46, 48); discretionary denial under § 314(a) (Prelim. Resp. 49–54; Reply 10; Sur-Reply 5); and alleged failure to name all real parties in interest under § 312(a)(2) (Prelim. Resp. 54–63; Reply 1–9; Sur-Reply 1–5; Sur-Sur-Reply 1–2).

IV. CONCLUSION

In light of the foregoing, the Petition fails to demonstrate a reasonable likelihood of prevailing on its assertions that claims 1, 2 and 4–12 of the '073 Patent would have been obvious under 35 U.S.C. § 103. Accordingly, we decline to institute trial on claims 1, 2 and 4–12 on the grounds of unpatentability asserted by Petitioner.

V. ORDER

For the foregoing reasons, it is ORDERED that the Petition for *inter partes* review is *denied*.

For Petitioner:

Trenton Ward
A. Grace Mills
FINNEGAN, HENDERSON, FARABOW, GARRETT & DUNNER, LLP
Trenton.ward@finnegan.com
Gracie.mills@finnegan.com

Ashraf A. Fawzy
Jung Hahm
UNIFIED PATENTS, LLC
afawzy@unifiedpatents.com
jung@unifiedpatents.com

For Patent Owner:

Leslie V. Payne
Christopher L. Limbacher
HEIM PAYNE & CHORUSH, LLP
lpayne@hpcllp.com
climbacher@hpcllp.com

UNITED STATES DISTRICT COURT
NORTHERN DISTRICT OF CALIFORNIA
SAN FRANCISCO DIVISION

MASTEROBJECTS, INC., *Plaintiff*, v. AMAZON.COM, INC., *Defendant*.	Civil Action No. 5:20-cv-08103-WHA JURY TRIAL DEMANDED

PLAINTIFF MASTEROBJECTS, INC'S PRELIMINARY INFRINGEMENT CONTENTIONS

In accordance with Section 2 of the Order Governing Proceedings – Patent Case (ECF 29), Plaintiff MasterObjects, Inc. ("MasterObjects" or "Plaintiff"), hereby: (1) provides its preliminary infringement contentions in the form of a chart—attached as Exhibit A—setting forth where in the accused instrumentalities each element of the asserted claims are found; (2) identifies the priority date (i.e., the earliest date of invention) for each asserted claim; and (3) provides an accompanying production which includes documents evidencing conception and reduction to practice for each claimed invention and copies of the file histories for each patent-in-suit:

I. Disclosure of a Preliminary Infringement Contentions Chart:

The patents-in-suit are (1) U.S. Patent No. 8,539,024 (the "'024 Patent"); (2) United States Patent No. 9,760,628 (the "'628 Patent"); (3) United States Patent No. 10,394,866 (the "'866 Patent"); and (4) United States Patent No. 10,311,073 (the "'073 Patent") (collectively the "Patents-in-Suit"). The accused instrumentalities are the social media platforms offered by Defendant Amazon.com, Inc. ("Amazon" or "Defendant") through, *e.g.*, its Amazon.com web application, through its Amazon mobile applications, including but not limited to, those for iOS and Android operating systems, and through any other Amazon desktop or mobile applications.

These accused instrumentalities include both client-side and server-side functionality used to process, send, receive, cache, and retrieve search results asynchronously. As described in Exhibit A and Plaintiff's complaint, the accused instrumentalities include the Amazon search features known as Advanced Search Looking Glass. Exhibit A sets forth representative examples showing where in the accused instrumentalities each element of each asserted claim is found. Amazon infringed, and is infringing, each claim in Exhibit A. Amazon's infringing activities constitute at least direct infringement under 35 U.S.C. § 271(a), literally or under the doctrine of equivalents. As alleged in MasterObjects' complaint, Amazon's infringement is willful.

If a claim of a Patent-in-Suit is not identified by MasterObjects in Exhibit A, then that claim is not presently asserted by MasterObjects against Amazon.

Pursuant to the Order Governing Proceedings – Patent Case, the infringement contentions hereby disclosed by MasterObjects are preliminary. *See* ECF 29, § 2, App'x A at row 1 & n. 4. Pursuant to the Order Governing Proceedings – Patent Case, the deadline to serve final infringement contentions is over eight months away. *See id.,* App'x A at row 15.

These disclosures, including Exhibit A, are based on the present state of MasterObjects' knowledge, without the benefit of any discovery. Further, MasterObjects' investigation is ongoing, and no *Markman* order has been entered in this action. MasterObjects reserves all rights to supplement, amend, and/or otherwise modify its infringement contentions.

The parties have not exchanged claim terms or proposed claim constructions, Defendant has not served its preliminary invalidity contentions and accompanying production, and the *Markman* hearing is over six months away. MasterObjects is not required to disclose claim construction positions at this time, and does not. These disclosures, inclusive of Exhibit A, should not be construed as setting forth MasterObjects' claim construction positions. To the extent Defendant asserts that a particular MasterObjects claim construction position is implied by these disclosures, including Exhibit A, MasterObjects denies and objects to any such assertion. MasterObjects reserves all rights to modify its claim construction positions.

II. Disclosure of the Priority Date:

The ’024 Patent was filed on February 6, 2012. The ’024 Patent is a continuation of U.S. Patent No. 8,112,529 (the ’529 Patent), which was filed on August 20, 2001. The asserted ’024 Patent claims are entitled to the benefit of the ’529 Patent’s filing date. The asserted claims of the ’024 Patent were conceived prior to the filing of the ’529 Patent. The asserted ’024 Patent claims were conceived no later than November 11, 2000. The earliest priority date currently claimed by MasterObjects for the asserted ’024 Patent claims is November 11, 2000.

The ’628 Patent was filed on September 16, 2013. The ’628 Patent is a continuation of the ’024 Patent, which is a continuation of the ’529 Patent. The ’529 Patent was filed on August 20, 2001. The asserted ’628 Patent claims are entitled to the benefit of the ’529 Patent’s filing date. The asserted claims of the ’628 Patent were conceived prior to the filing of the ’529 Patent. The asserted ’628 Patent claims were conceived no later than November 11, 2000. The earliest priority date currently claimed by MasterObjects for the asserted ’628 Patent claims is November 11, 2000.

The ’866 Patent was filed on December 22, 2016. The ’866 Patent is a continuation of the ’628 Patent, which is a continuation of the ’024 Patent, which is a continuation of the ’529 Patent. The ’529 Patent was filed on August 20, 2001. The asserted ’866 Patent claims are entitled to the benefit of the ’529 Patent’s filing date. The asserted claims of the ’866 Patent were conceived prior to the filing of the ’529 Patent. The asserted ’866 Patent claims were conceived no later than November 11, 2000. The earliest priority date currently claimed by MasterObjects for the asserted ’866 Patent claims is November 11, 2000.

The ’073 Patent was filed on February 17, 2017. The ’073 Patent is related to the ’529 Patent as follows: the ’073 Patent is a continuation of U.S. Application No. 12/176,984; U.S. Application No. 12/176,984 is a continuation-in-part of U.S. Patent No. 7,752,326; U.S. Patent No. 7,752,326 is a continuation-in-part of the ’529 Patent. The ’529 Patent was filed on August 20, 2001. The asserted ’073 Patent claims are entitled to the benefit of the ’529 Patent’s filing date. The asserted claims of the ’073 Patent were conceived prior to the filing of the ’529 Patent. The

asserted ’073 Patent claims were conceived no later than November 11, 2000. The earliest priority date currently claimed by MasterObjects for the asserted ’073 Patent claims is November 11, 2000.

These disclosures are based on the present state of MasterObjects’ knowledge. Further, MasterObjects’ investigation is ongoing. MasterObjects reserves all rights to modify the positions taken in these initial disclosures.

III. Disclosure of Accompanying Production:

These disclosures include an accompanying document production that includes documents evidencing conception and reduction to practice for each claimed invention and copies of the file histories for each Patent-in-Suit. The accompanying production is subject to and without waiving the objections and reservations set forth herein. The Bates number ranges for the accompanying production are: MOBJECTS_0000001 – MOBJECTS_0000943; MOBJECTS_0044288 – MOBJECTS_0045474; and MO_000001 – MO_002349.

Masterobjects objects to the production of any documents protected by the attorney-client privilege, the work-product doctrine, or any other immunities from discovery.

In producing the accompanying documents, MasterObjects does not admit or concede the relevancy, materiality, authenticity, or admissibility as evidence of any of these documents. All objections to the use, at trial or otherwise, of any document produced are hereby expressly reserved.

MasterObjects makes these disclosures without the benefit of discovery. Further, MasterObjects’ investigation is ongoing. MasterObjects produces these documents without prejudice to its right to produce additional documents after considering documents obtained and reviewed throughout discovery and further investigation.

IV. Confidentiality.

Pursuant to Section “Protective Order” of the Order Governing Proceedings – Patent Case, MasterObjects designated portions of these disclosures, including Exhibit A and various documents in the accompanying production, “confidential.” MasterObjects identified the material designated confidential by marking it “CONFIDENTIAL,” “HIGHLY-CONFIDENTIAL ATTORNEYS’ EYES ONLY,” or with some other similar confidentiality marker. For the avoidance of doubt,

under Section "Protective Order" of the Order Governing Proceedings – Patent Case, a document marked "CONFIDENTIAL" is to be treated in the same way as a document marked "HIGHLY-CONFIDENTIAL ATTORNEYS' EYES ONLY." Pursuant to the Order Governing Proceedings – Patent Case, given no Protective Order has yet been issued by the Court, disclosure of the designated materials is limited to Amazon's outside attorneys of record and the employees of such outside attorneys.

Dated: May 15, 2020

Respectfully submitted,

/s/ Darrell R. Atkinson

Spencer Hosie, *pro hac vice*,
(CA Bar No. 101777)
shosie@hosielaw.com
Diane S. Rice, *pro hac vice*,
(CA Bar No. 118303)
drice@hosielaw.com
Brandon C. Martin, *pro hac vice*,
(CA Bar No. 269624)
bmartin@hosielaw.com
Darrell Rae Atkinson, *pro hac vice*,
(CA Bar No. 280564)
datkinson@hosielaw.com
Francesca M.S. Germinario, *pro hac vice*,
(CA Bar No. 326208)
fgerminario@hosielaw.com
HOSIE RICE LLP
600 Montgomery St., 34th Floor
San Francisco, CA 94111
415.247.6000
Fax: 415.247.6001

Leslie V. Payne
(TX State Bar No. 00784736)
lpayne@hpcllp.com
Alden G. Harris, *pro hac vice*,
(TX State Bar No. 24083138)
aharris@hpcllp.com
HEIM, PAYNE & CHORUSH, LLP
1111 Bagby St., Ste. 2100
Houston, Texas 77002
Telephone: (713) 221-2000
Facsimile: (713) 221-2021

CERTIFICATE OF SERVICE

I, Francesca M.S. Germinario, am a citizen of the United States of America and am employed in the County of San Francisco, State of California. I am over the age of 18 years and am not a party to the within action. My business address is Hosie Rice LLP, Transamerica Pyramid, 34th Floor, 600 Montgomery Street, San Francisco, California, 94111.

On May 15, 2020, I served the following:

- **PLAINTIFF MASTEROBJECTS, INC'S PRELIMINARY INFRINGEMENT CONTENTIONS (INCLUSIVE OF EXHIBIT A)**

via email, and the following by electronic file transfer (DropBox):

- **AN ACCOMPANYING DOCUMENT PRODUCTION**

at South San Francisco, California, addressed to the following parties:

Douglas E. Lumish
Jeffrey G. Homrig
Clara Wang
Latham & Watkins LLP
140 Scott Drive
Menlo Park, CA 94025
Doug.Lumish@lw.com
Jeff.Homrig@lw.com
Clara.Wang@lw.com

Joseph H. Lee
Latham & Watkins LLP
650 Town Center Drive, 20th Floor
Costa Mesa, CA 92626
Joseph.Lee@lw.com

Tiffany C. Weston
Rachel Weiner Cohen
Latham & Watkins LLP
555 Eleventh Street, NW, Suite 1000
Washington, D.C. 20004
Tiffany.Weston@lw.com
Rachel.Cohen@lw.com

Paul Weinand
Latham & Watkins LLP
200 Clarendon Street
Boston, MA 02116
Paul.Weinand@lw.com

Paige Arnette Amstutz
Scott Douglas & McConnico LLP
303 Colorado Street, Suite 2400
Austin, TX 78701
pamstutz@scottdoug.com

Attorneys for Defendant Amazon.com, Inc.

I declare under penalty of perjury under the laws of the United States of America that the foregoing is true and correct.

DATED: May 15, 2020

/s/ Francesca M.S. Germinario
Francesca M.S. Germinario

From: The Diligence Team <diligence.team@litfin.co>
To: Hosie Rice LLC Litigation Support Paralegal <paralegal@hosierice.com>
Date: July 22, 2024
Subject: MasterObjects, Inc., Proposed Patent Lawsuits

Dear Litigation Support Team,

We are currently conducting due diligence for potential investment in a series of lawsuits involving MasterObjects, Inc., specifically related to U.S. Patent Nos. 8,539,024, 9,760,628, 10,311,073, and 10,394,866 (the "Patents-in-Suit").

To ensure we make an informed decision, we need to gather detailed information regarding the financial and economic status of MasterObjects, Inc. Specifically, we would like to understand:

1. **Financial Solvency:** Can you provide an overview of MasterObjects' current financial health, including their profitability, liquidity, and any outstanding debts? Are they generating sufficient revenue to cover their operational expenses?
2. **Risk of Asset Seizure:** Are there any known risks of MasterObjects' assets being seized or otherwise compromised due to existing financial obligations, liens, or pending litigation that could affect their ability to sustain a lengthy legal battle?

Additionally, we have some further questions:

1. **Ongoing Products and Revenue Streams:** Does MasterObjects, Inc. have any ongoing products or services that could be targeted through counterclaims in litigation? If so, what are these products and how significant are they to their overall revenue stream?
2. **Contingency Fee Agreement:** Is there a contingency fee agreement in place with their current legal representation? If not, how are their legal fees structured and are there any significant financial commitments that could impact their litigation strategy?
3. **Previous Litigation:** Have the Patents-in-Suit been previously litigated? If so, can you provide details on the nature of the lawsuits, the parties involved, and the outcomes? Were the settlements or judgments favorable, and did they have any long-term impact on MasterObjects' business operations?

Your prompt response to these inquiries will be greatly appreciated and will aid in our decision-making process.

Best regards,

The Diligence Team
Litigation Finance Group
Email: diligence.team@litfin.co
Phone: (555) 123-4567

From: Hosie Rice LLC Litigation Support Paralegal <paralegal@hosierice.com>
To: The Diligence Team <diligence.team@litfin.co>
Date: July 23, 2024
Subject: Re: MasterObjects, Inc., Proposed Patent Lawsuits

Dear Diligence Team,

Thank you for your email. Please find our responses below:

> To ensure we make an informed decision, we need to gather detailed information regarding the financial and economic status of MasterObjects, Inc. Specifically, we would like to understand:
>
> 1. **Financial Solvency:** Can you provide an overview of MasterObjects' current financial health, including their profitability, liquidity, and any outstanding debts? Are they generating sufficient revenue to cover their operational expenses?

MasterObjects, Inc. is primarily a licensing and holding company without their own products. They generate revenue through licensing agreements and have been profitable and solvent over the past five years. Their financial statements indicate a strong liquidity position with no significant outstanding debts. Over the last fiscal year, they reported a net income of $12 million, with a consistent revenue stream from multiple long-term licensing agreements. The company has maintained a current ratio above 2.5, highlighting its ability to cover short-term liabilities comfortably. Moreover, their audited financial reports reflect prudent cash management practices, ensuring adequate reserves for both operational and legal expenses.

> 2. **Risk of Asset Seizure:** Are there any known risks of MasterObjects' assets being seized or otherwise compromised due to existing financial obligations, liens, or pending litigation that could affect their ability to sustain a lengthy legal battle?

There are no known risks of asset seizure for MasterObjects, Inc. They do not have any significant financial obligations, liens, or pending litigation that would compromise their assets or affect their ability to sustain prolonged legal proceedings.

> Additionally, we have some further questions:
>
> 1. **Ongoing Products and Revenue Streams:** Does MasterObjects, Inc. have any ongoing products or services that could be targeted through counterclaims in litigation? If so, what are these products and how significant are they to their overall revenue stream?

MasterObjects, Inc. is a licensing and holding company without any physical products of their own. Therefore, there are no products that could be targeted through counterclaims in litigation. Their revenue is entirely derived from licensing the Patents-in-Suit and other intellectual property.

> 2. **Contingency Fee Agreement:** Is there a contingency fee agreement in place with their current legal representation? If not, how are their legal fees structured and are there any significant financial commitments that could impact their litigation strategy?

We are currently gathering detailed information regarding the contingency fee agreement and the overall structure of their legal fees. We will get back to you with this information shortly.

From: The Diligence Team <diligence.team@litfin.co>
To: Hosie Rice LLC Litigation Support Paralegal <paralegal@hosierice.com>
Date: July 24, 2024
Subject: Re: MasterObjects, Inc., Proposed Patent Lawsuits

Dear Litigation Support Team,

Thank you for your prompt and detailed responses. The information provided is very helpful.

We have a few additional questions to further complete our due diligence:

1. Has the patent been granted or challenged in any other jurisdictions?
2. Does the patent claim priority back to a previous patent, and was that patent in the same language as the currently asserted patent?

We appreciate your continued assistance in this matter.

Best regards,

The Diligence Team
Litigation Finance Group
Email: diligence.team@litfin.co
Phone: (555) 123-4567

From: Hosie Rice LLC Litigation Support Paralegal <paralegal@hosierice.com>
To: The Diligence Team <diligence.team@litfin.co>
Date: July 25, 2024
Subject: Re: MasterObjects, Inc., Proposed Patent Lawsuits

Dear Diligence Team,

Thank you for your follow-up questions. Please find the responses below, including the complete answers to the previously incomplete inquiries:

> We are currently conducting due diligence for potential investment in a series of lawsuits involving MasterObjects, Inc., specifically related to U.S. Patent Nos. 8,539,024, 9,760,628, 10,311,073, and 10,394,866 (the "Patents-in-Suit").
>
> To ensure we make an informed decision, we need to gather detailed information regarding the financial and economic status of MasterObjects, Inc. Specifically, we would like to understand:
>
> 1. **Financial Solvency:** Can you provide an overview of MasterObjects' current financial health, including their profitability, liquidity, and any outstanding debts? Are they generating sufficient revenue to cover their operational expenses?
> 2. **Risk of Asset Seizure:** Are there any known risks of MasterObjects' assets being seized or otherwise compromised due to existing financial obligations, liens, or pending litigation that could affect their ability to sustain a lengthy legal battle?
>
> Additionally, we have some further questions:
>
> 1. **Ongoing Products and Revenue Streams:** Does MasterObjects, Inc. have any ongoing products or services that could be targeted through counterclaims in litigation? If so, what are these products and how significant are they to their overall revenue stream?
> 2. **Contingency Fee Agreement:** Is there a contingency fee agreement in place with their current legal representation? If not, how are their legal fees structured and are there any significant financial commitments that could impact their litigation strategy?
> 3. **Previous Litigation:** Have the Patents-in-Suit been previously litigated? If so, can you provide details on the nature of the lawsuits, the parties involved, and the outcomes? Were the settlements or judgments favorable, and did they have any long-term impact on MasterObjects' business operations?

Complete Responses:

> **Contingency Fee Agreement:** Is there a contingency fee agreement in place with their current legal representation? If not, how are their legal fees structured and are there any significant financial commitments that could impact their litigation strategy?

MasterObjects, Inc. does not have a contingency fee agreement with their current legal representation. Their legal fees are structured on an hourly basis, with an upfront retainer fee to cover initial costs. The company has allocated sufficient financial resources to cover these legal fees, ensuring that there are no significant financial commitments that could impact their litigation strategy.

> **Previous Litigation:** Have the Patents-in-Suit been previously litigated? If so, can you provide details on the nature of the lawsuits, the parties involved, and the outcomes? Were the settlements or judgments favorable, and did they have any long-term impact on MasterObjects' business operations?

The Patents-in-Suit have been previously litigated in two cases involving midsized e-commerce platforms. Both lawsuits were settled favorably for MasterObjects, resulting in substantial settlement amounts. These settlements have positively impacted MasterObjects' financial position, and there were no adverse long-term impacts on their business operations.

New Responses:

Has the patent been granted or challenged in any other jurisdictions?

The Patents-in-Suit have not been granted in any other jurisdictions. There was an application filed in the Netherlands, but it was abandoned before grant.

Does the patent claim priority back to a previous patent, and was that patent in the same language as the currently asserted patent?

Yes, the Patents-in-Suit claim priority to an earlier filed U.S. patent. The earlier patent was also filed in English, maintaining consistency in the language across the priority chain.

We hope this information is helpful. Please let us know if you have any further questions or require additional details.

Best regards,

Hosie Rice LLC Litigation Support Paralegal
Email: paralegal@hosierice.com
Phone: (555) 987-6543

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.*

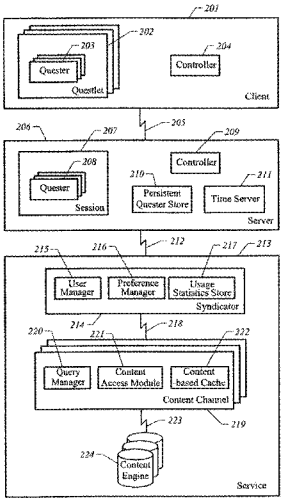
(Continued)

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?ls=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

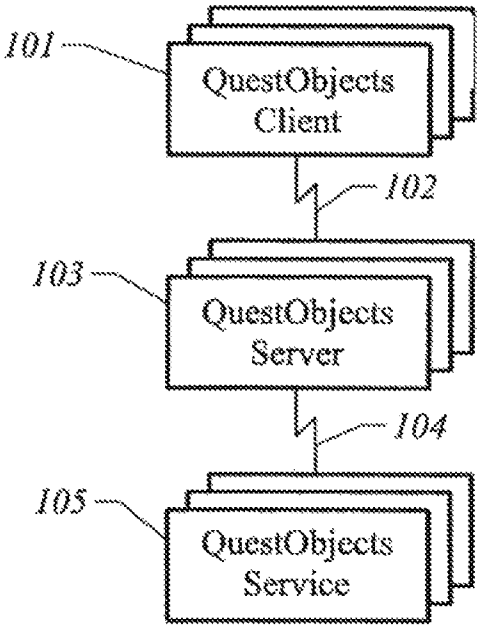

101
QuestObjects Client
102
103
QuestObjects Server
104
105
QuestObjects Service

FIG. 1

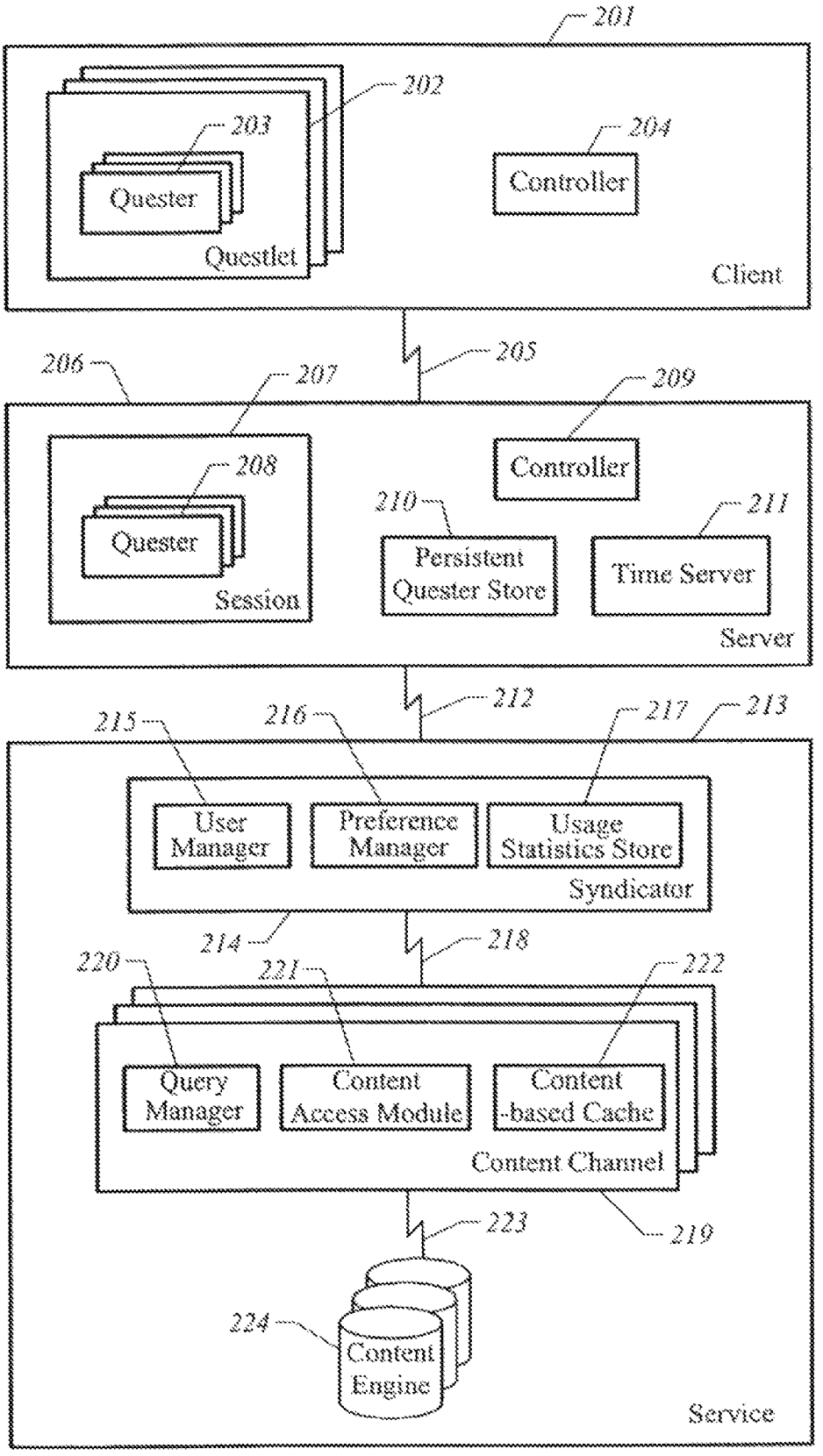

201
202
203
Quester
Questlet
204
Controller
Client
205
206
207
209
208
Quester
Session
Controller
210
211
Persistent Quester Store
Time Server
Server
212
215
216
217
213
User Manager
Preference Manager
Usage Statistics Store
Syndicator
214
218
220
221
222
Query Manager
Content Access Module
Content -based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

FIG. 2

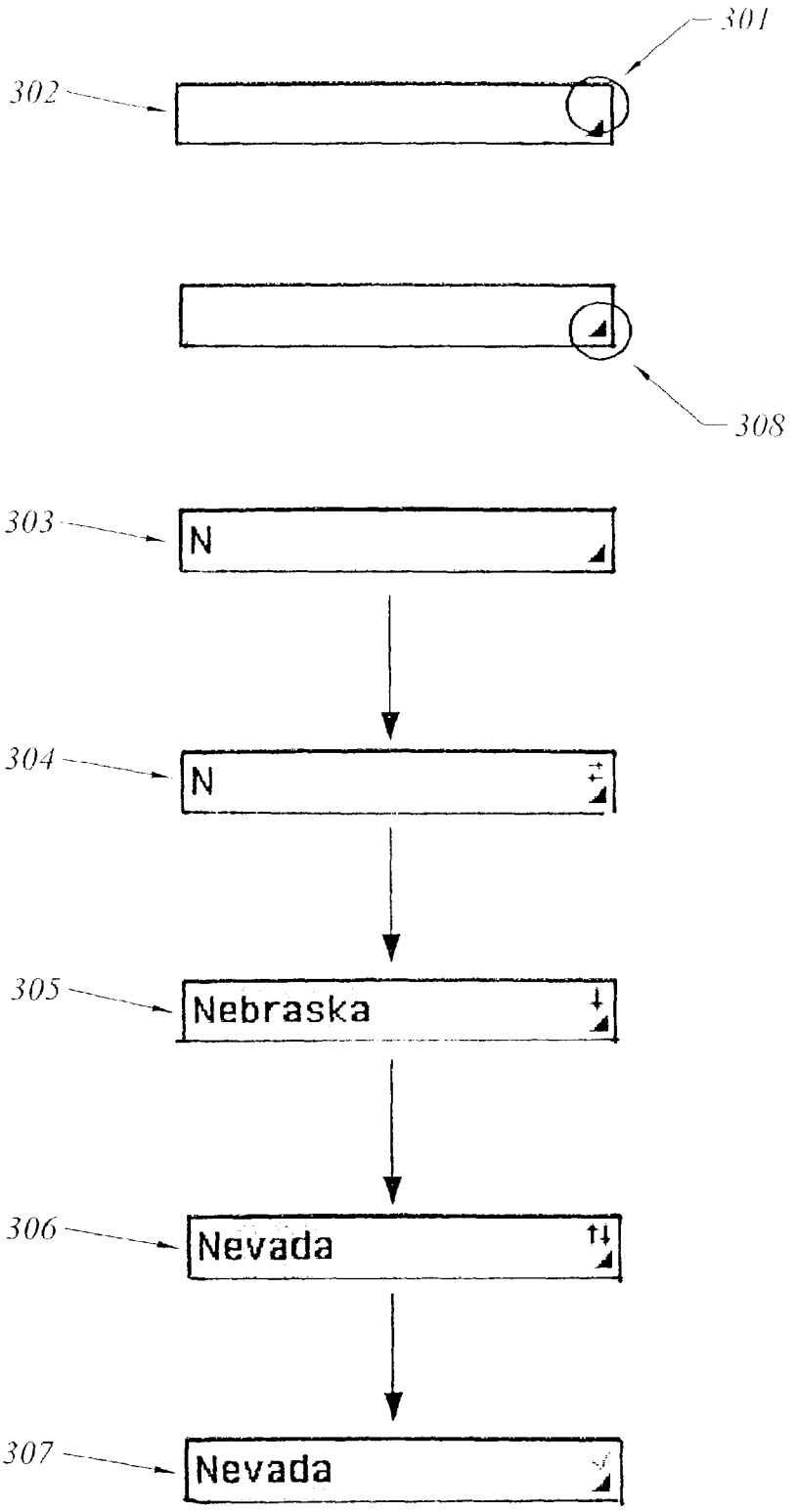


FIG. 3A

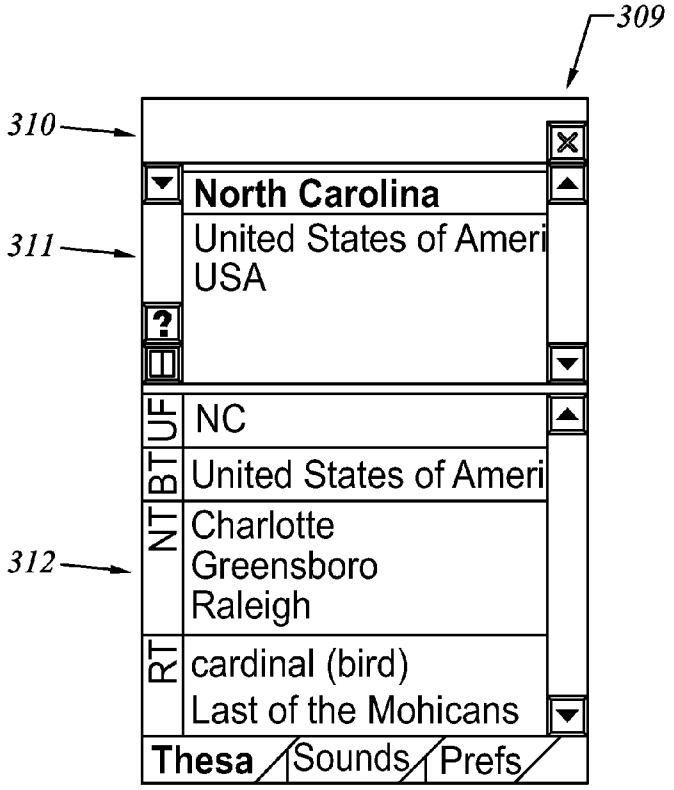

309
310
311
312
North Carolina
United States of Ameri
USA
UF NC
BT United States of Ameri
NT Charlotte
Greensboro
Raleigh
RT cardinal (bird)
Last of the Mohicans
Thesa
Sounds
Prefs

FIG. 3B

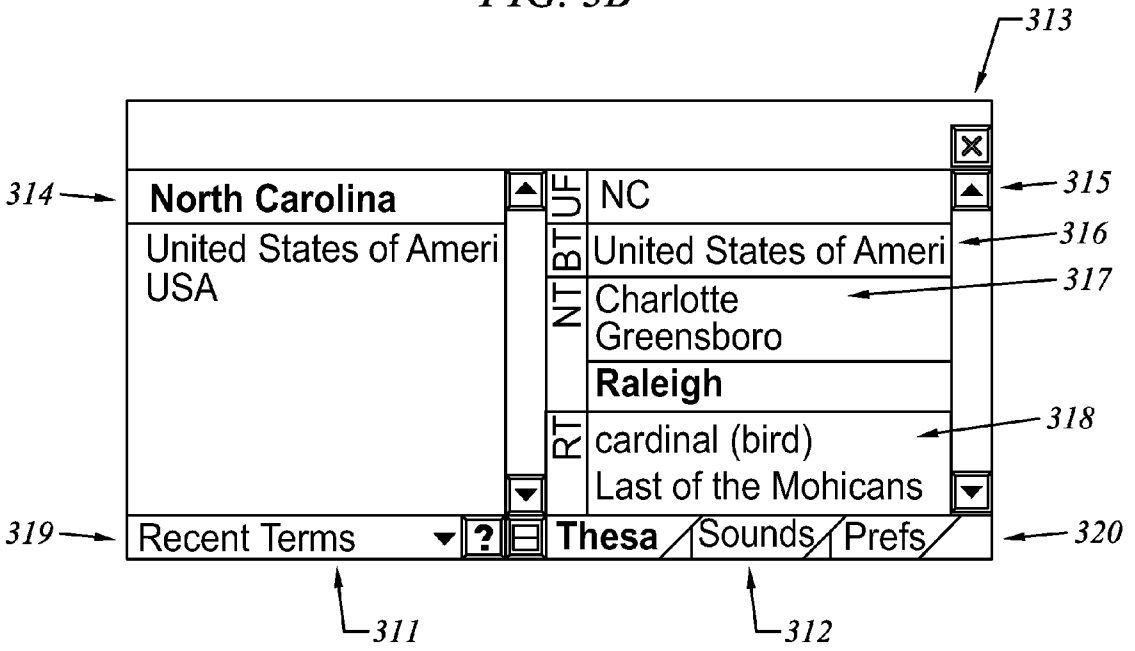

313
314
315
316
317
318
319
320
311
312
North Carolina
United States of Ameri
USA
UF NC
BT United States of Ameri
NT Charlotte
Greensboro
Raleigh
RT cardinal (bird)
Last of the Mohicans
Recent Terms
Thesa
Sounds
Prefs

FIG. 3C

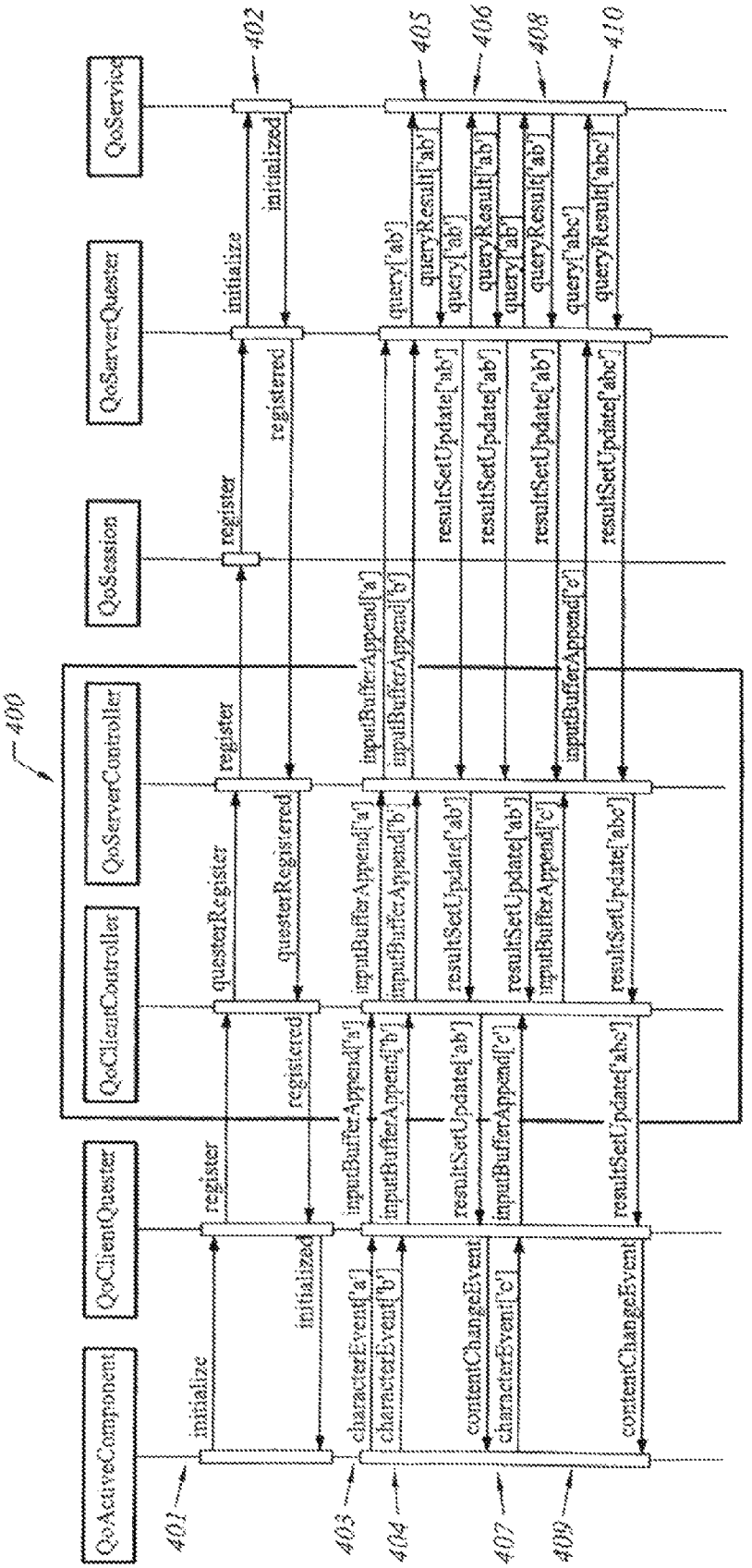

QoActiveComponent
QoClientQuester
QoClientController
QoServerController
QoSession
QoServerQuester
QoService
400
401
402
403
404
405
406
407
408
409
410
initialize
initialized
register
registered
questerRegister
questerRegistered
inputBufferAppend['a']
inputBufferAppend['b']
inputBufferAppend['c']
resultSetUpdate['ab']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

FIG. 4

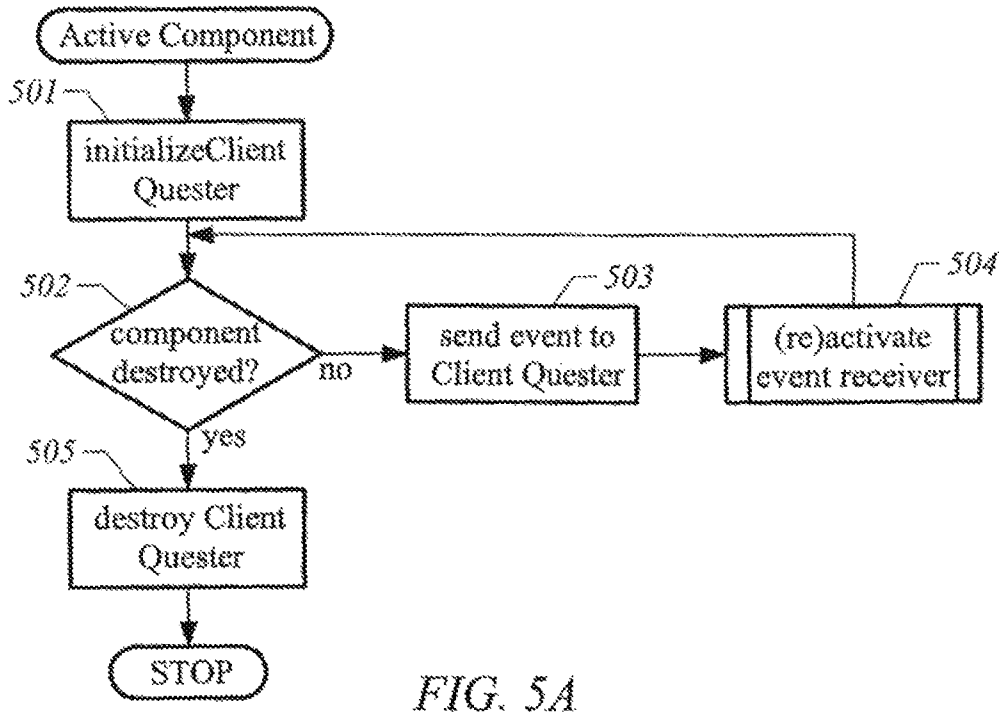

Active Component
501
initializeClient Quester
502
component destroyed?
no
503
send event to Client Quester
504
(re)activate event receiver
yes
505
destroy Client Quester
STOP

FIG. 5A

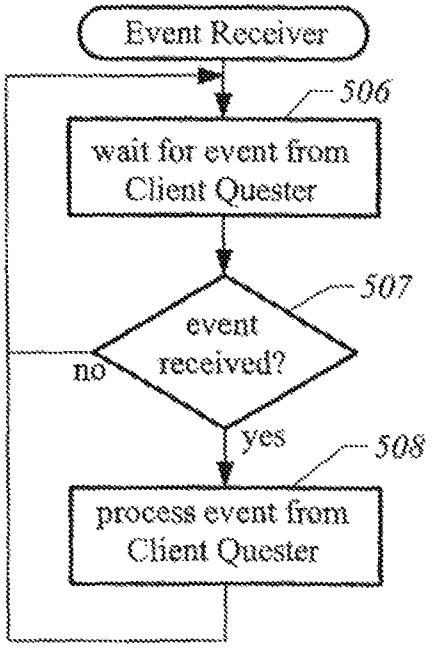

Event Receiver
506
wait for event from Client Quester
507
event received?
no
yes
508
process event from Client Quester

FIG. 5B

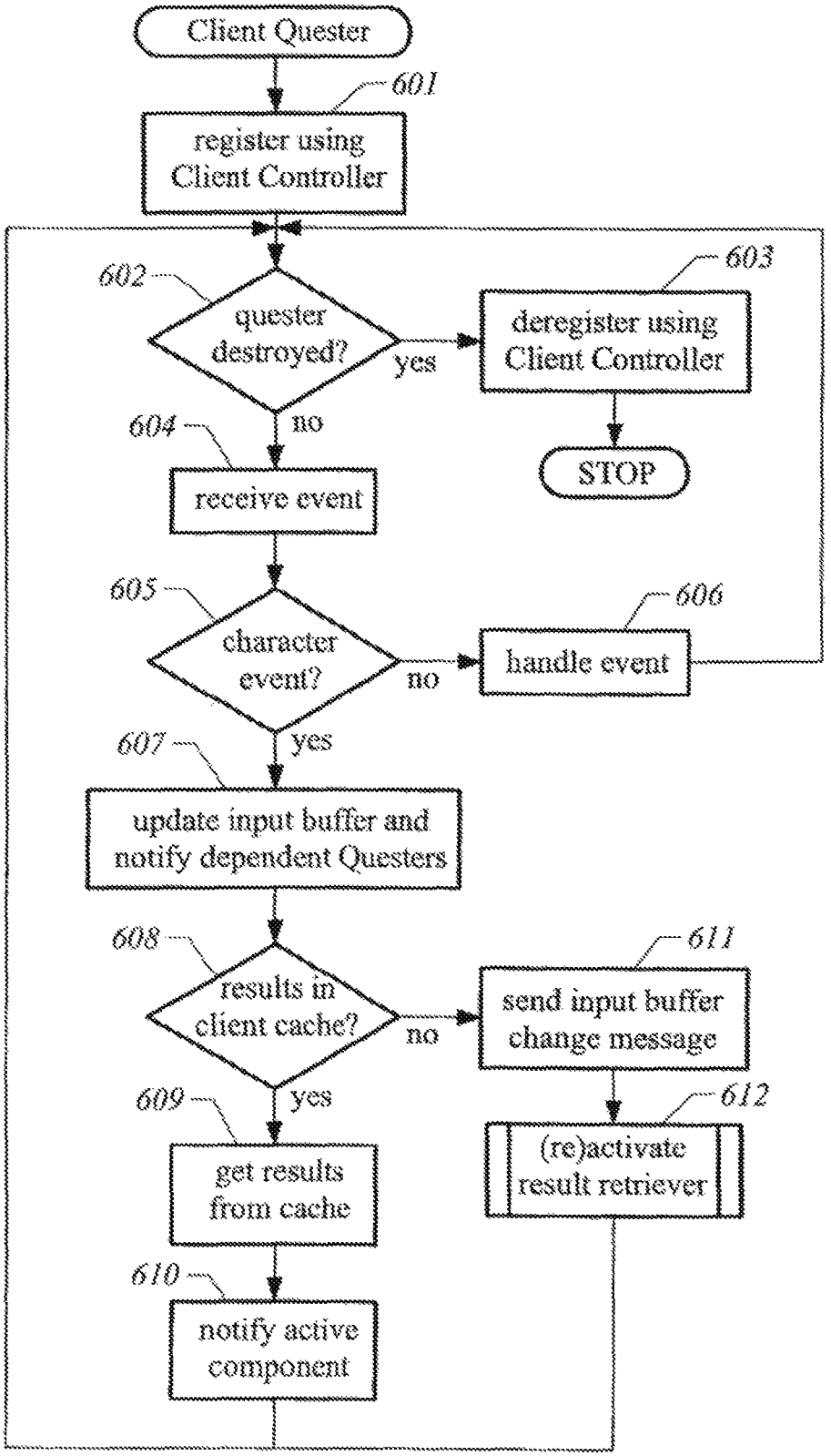

Client Quester
601
register using Client Controller
602
quester destroyed?
yes
603
deregister using Client Controller
STOP
no
604
receive event
605
character event?
no
606
handle event
yes
607
update input buffer and notify dependent Questers
608
results in client cache?
no
611
send input buffer change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

FIG. 6A

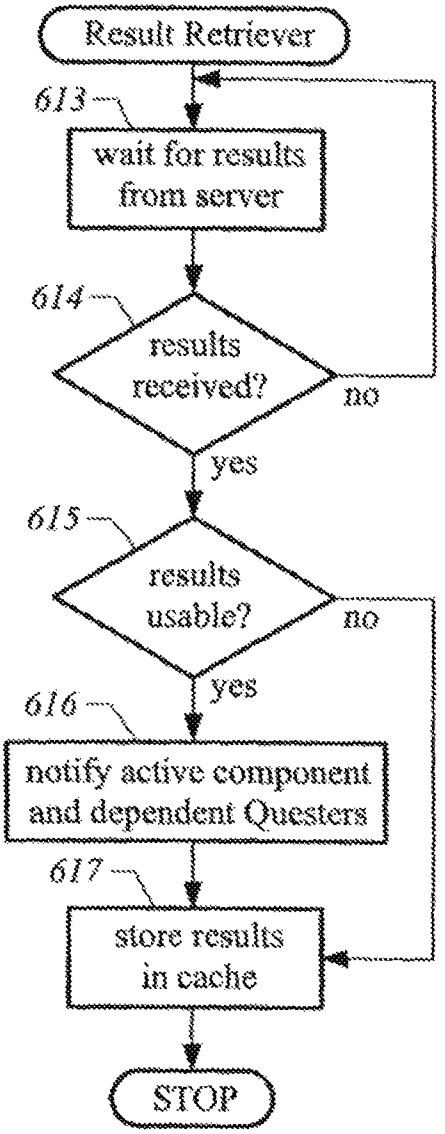

Result Retriever
613
wait for results from server
614
results received?
no
yes
615
results usable?
no
yes
616
notify active component and dependent Questers
617
store results in cache
STOP

FIG. 6B

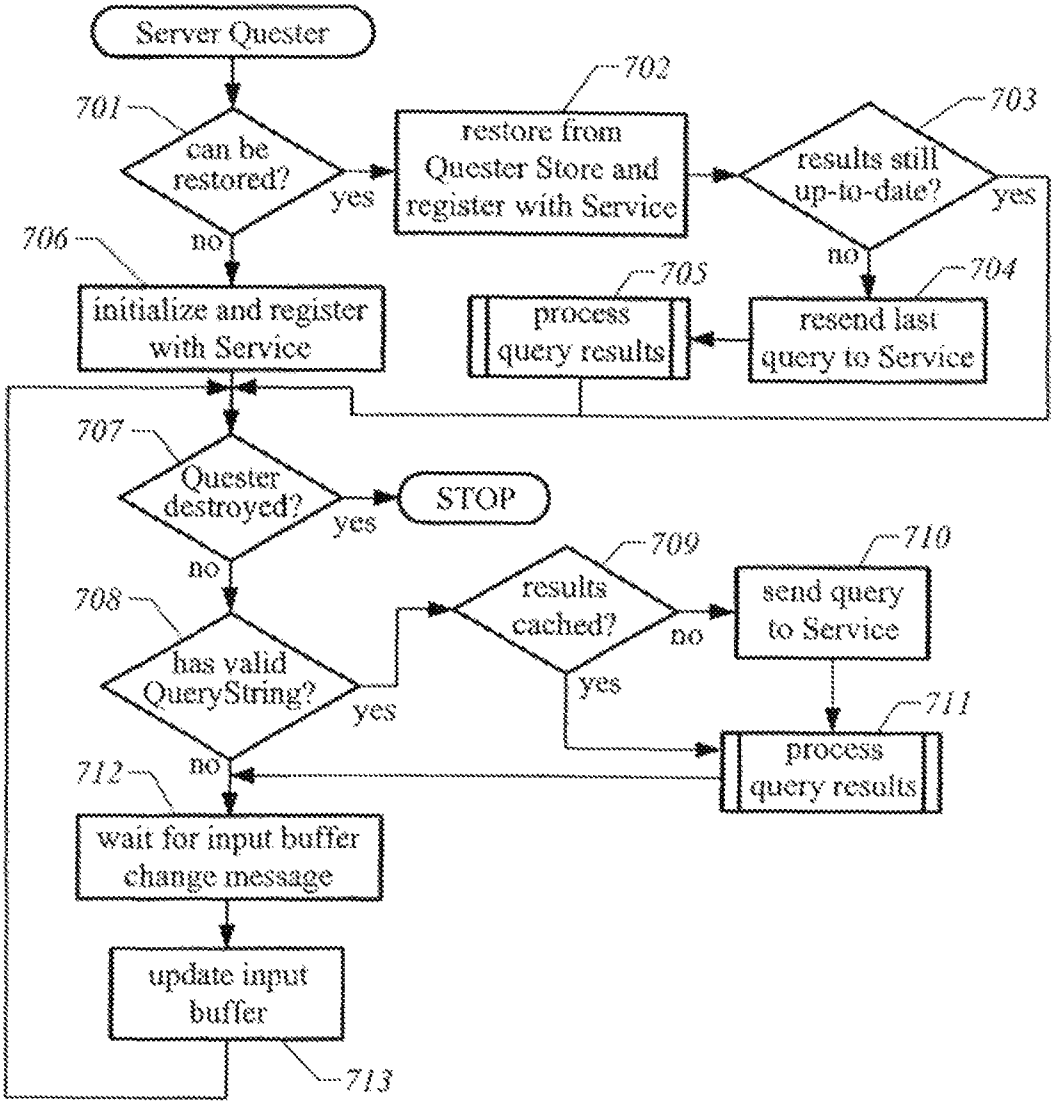

Server Quester
701
can be restored?
yes
no
702
restore from Quester Store and register with Service
703
results still up-to-date?
yes
no
704
resend last query to Service
705
process query results
706
initialize and register with Service
707
Quester destroyed?
yes
STOP
no
708
has valid QueryString?
yes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

FIG. 7A

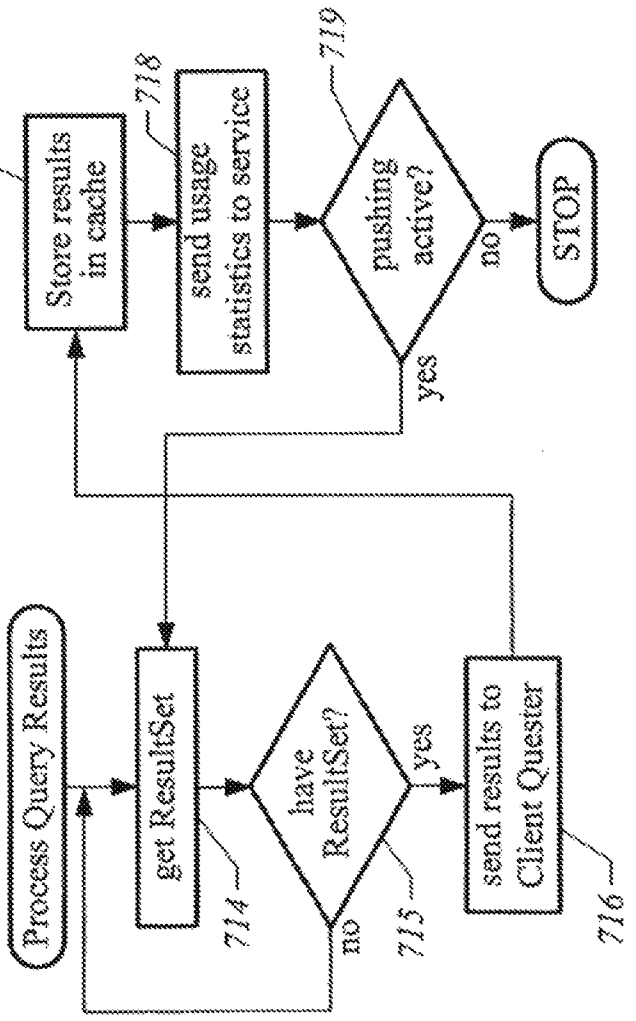

Process Query Results
get ResultSet
714
have ResultSet?
713
yes
no
send results to Client Quester
716
Store results in cache
717
send usage statistics to service
718
pushing active?
719
yes
no
STOP

FIG. 7B

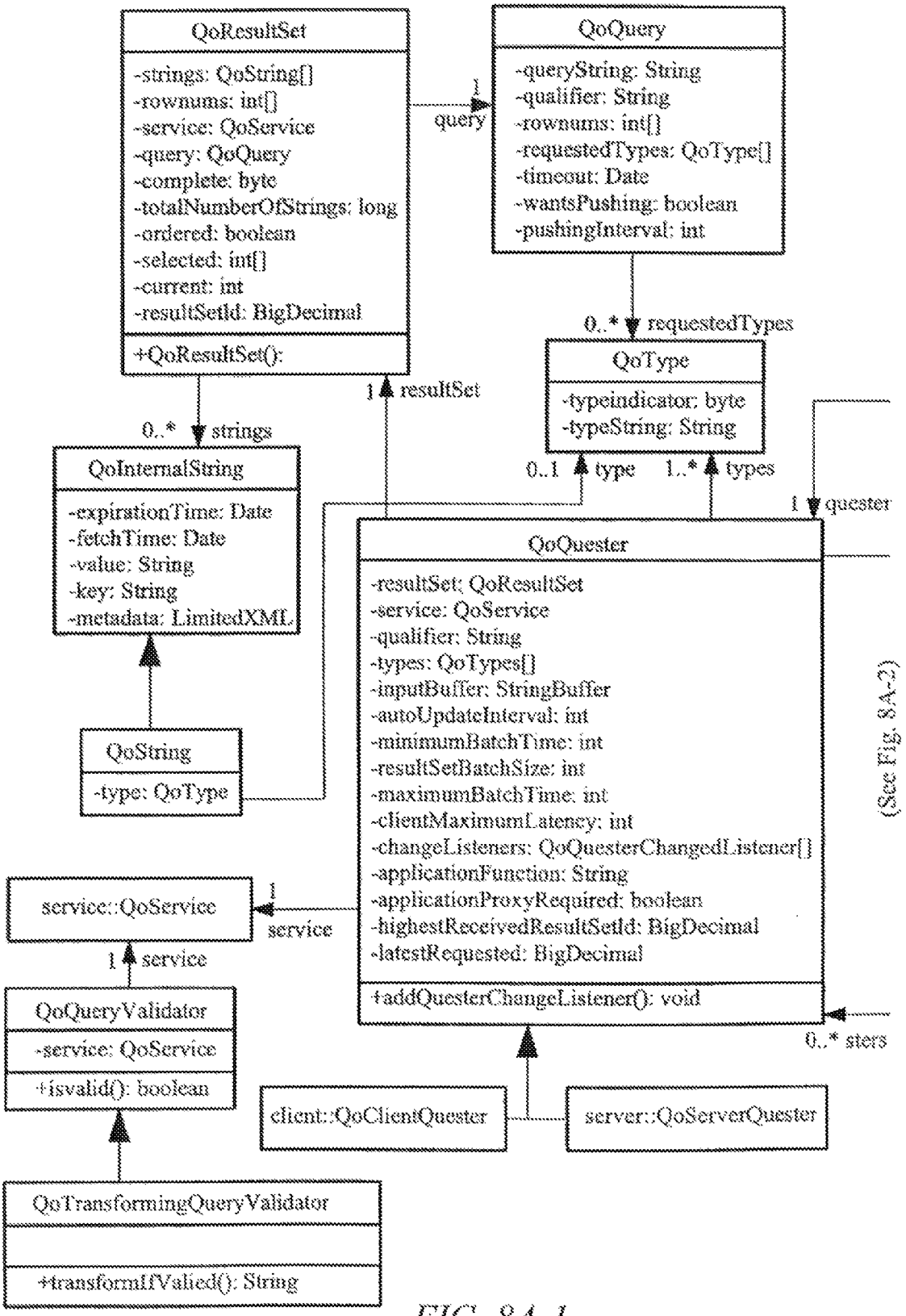

QoResultSet
-strings: QoString[]
-rownums: int[]
-service: QoService
-query: QoQuery
-complete: byte
-totalNumberOfStrings: long
-ordered: boolean
-selected: int[]
-current: int
-resultSetId: BigDecimal
+QoResultSet():
QoQuery
-queryString: String
-qualifier: String
-rownums: int[]
-requestedTypes: QoType[]
-timeout: Date
-wantsPushing: boolean
-pushingInterval: int
1
query
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
0..* sters
QoString
-type: QoType
service::QoService
1
service
1 service
QoQueryValidator
-service: QoService
+isvalid(): boolean
client::QoClientQuester
server::QoServerQuester
QoTransformingQueryValidator
+transformIfValied(): String
(See Fig. 8A-2)

FIG. 8A-1

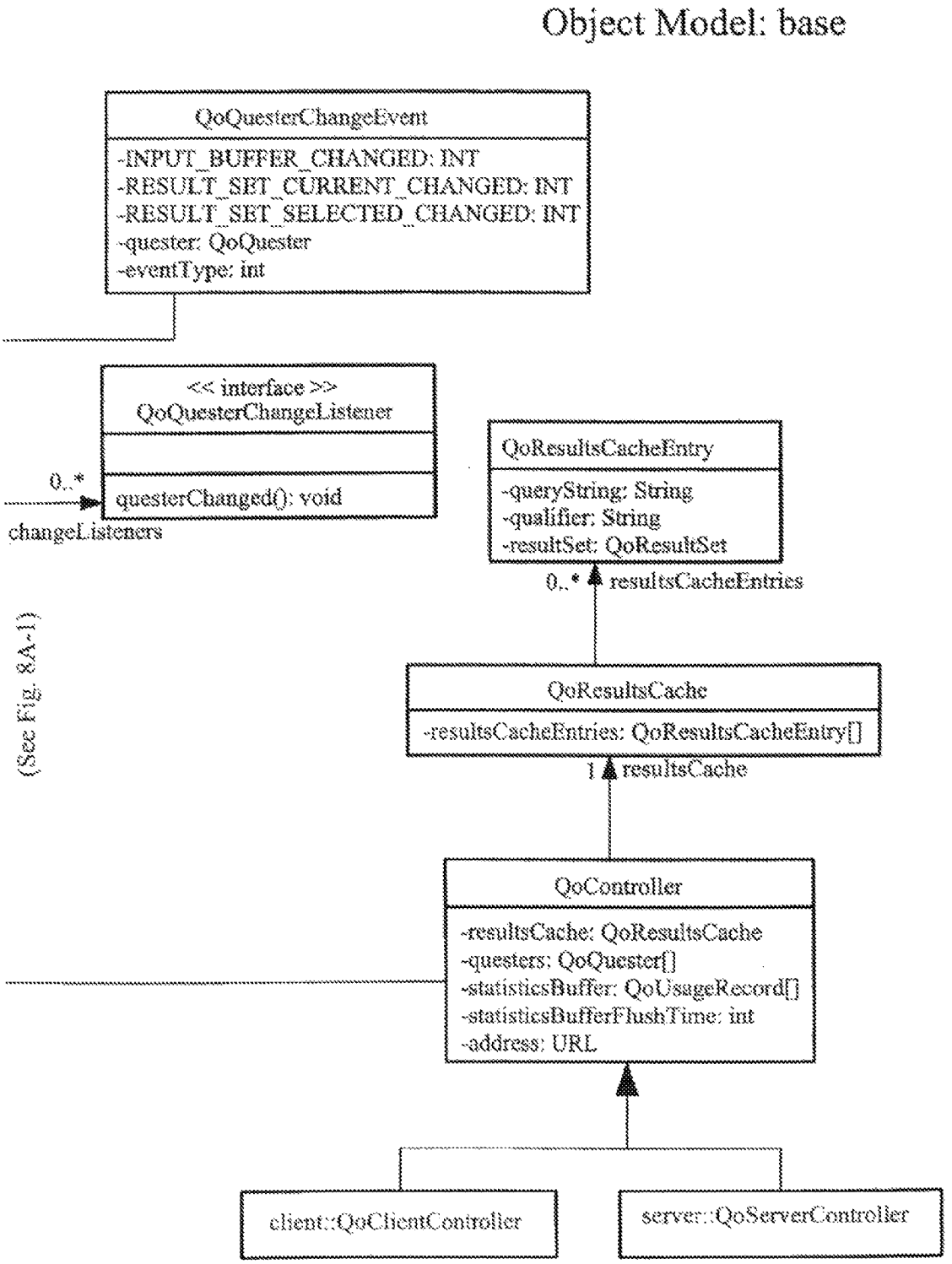

Object Model: base
QoQuesterChangeEvent
-INPUT_BUFFER_CHANGED: INT
-RESULT_SET_CURRENT_CHANGED: INT
-RESULT_SET_SELECTED_CHANGED: INT
-quester: QoQuester
-eventType: int
<< interface >>
QoQuesterChangeListener
questerChanged(): void
0..*
changeListeners
QoResultsCacheEntry
-queryString: String
-qualifier: 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

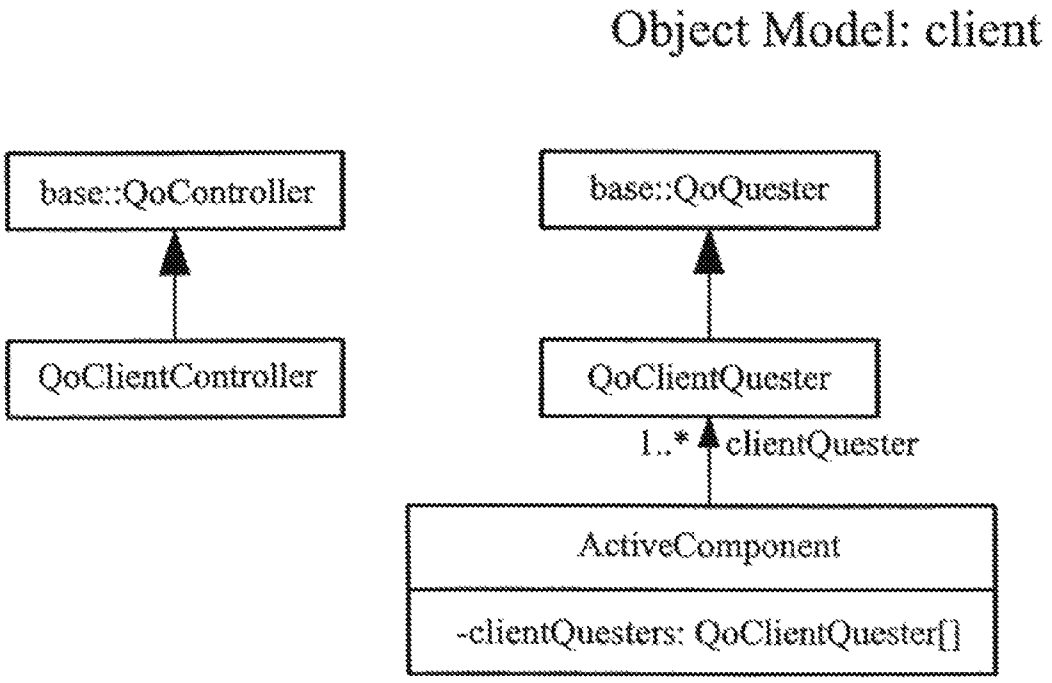

Object Model: client
base::QoController
QoClientController
base::QoQuester
QoClientQuester
1..*
clientQuester
ActiveComponent
-clientQuesters: QoClientQuester[]

FIG. 8B

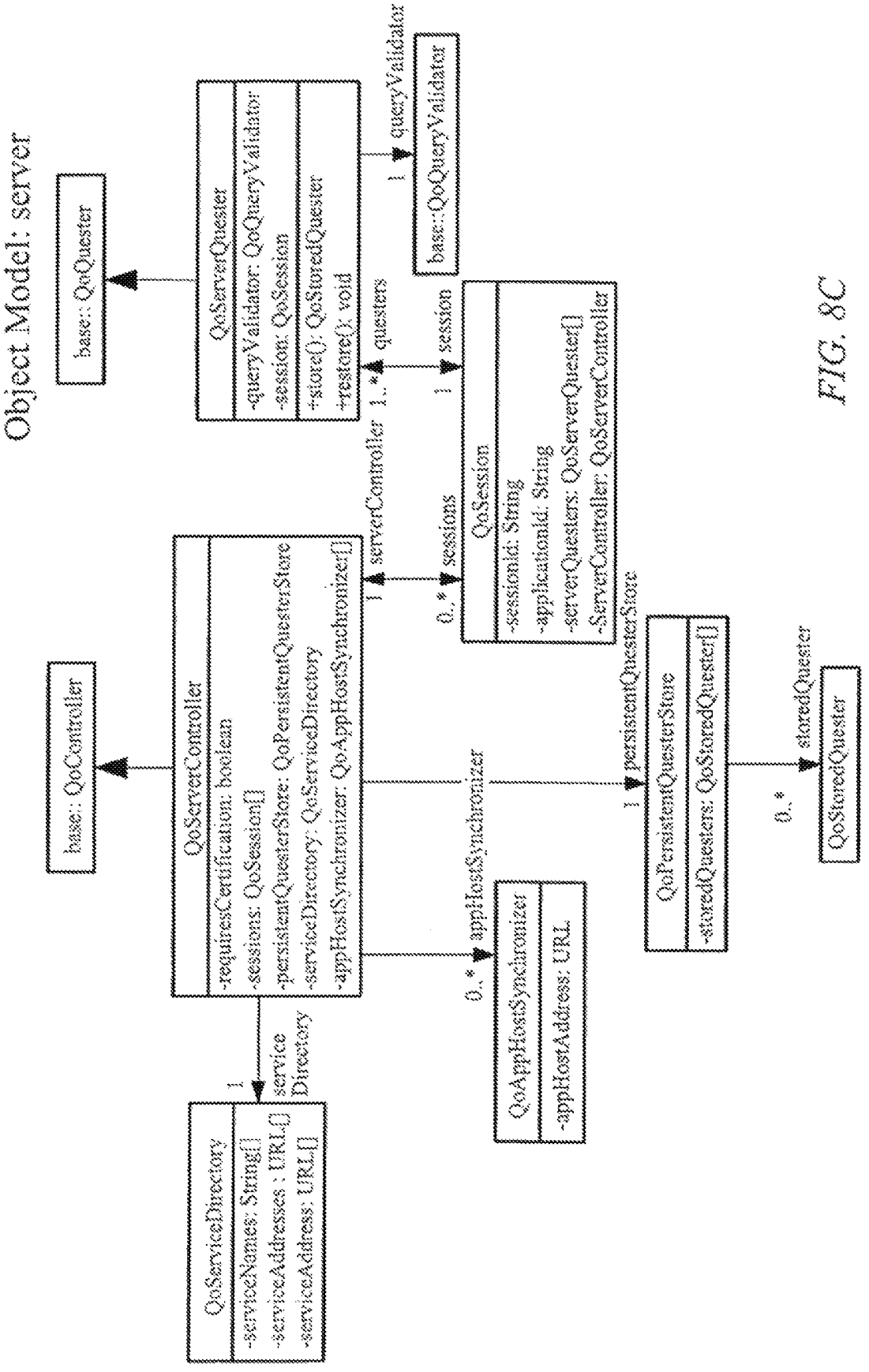

Object Model: server
base:: QoQuester
QoServerQuester
-queryValidator: QoQueryValidator
-session: QoSession
+store(): QoStoredQuester
+restore(): void
queryValidator
1
base::QoQueryValidator
1..* questers
1 session
1 serverController
base:: QoController
QoServerController
-requiresCertification: boolean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

FIG. 8C

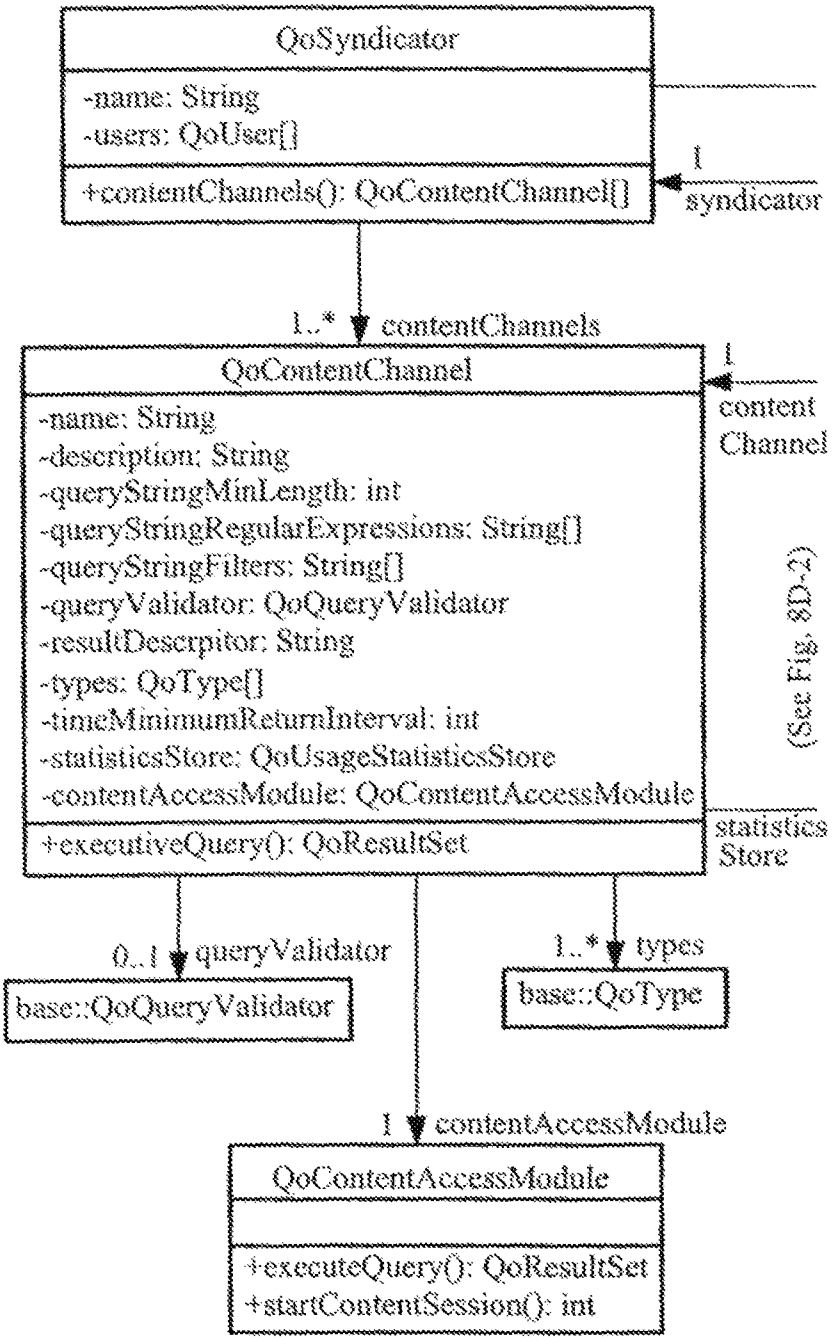

QoSyndicator
-name: String
-users: QoUser[]
+contentChannels(): QoContentChannel[]
1
syndicator
1..* contentChannels
QoContentChannel
1
content
Channel
-name: String
~description: String
-queryStringMinLength: int
-queryStringRegularExpressions: String[]
-queryStringFilters: String[]
-queryValidator: QoQueryValidator
-resultDescrpitor: String
-types: QoType[]
-timeMinimumReturnInterval: int
-statisticsStore: QoUsageStatisticsStore
-contentAccessModule: QoContentAccessModule
+executiveQuery(): QoResultSet
(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

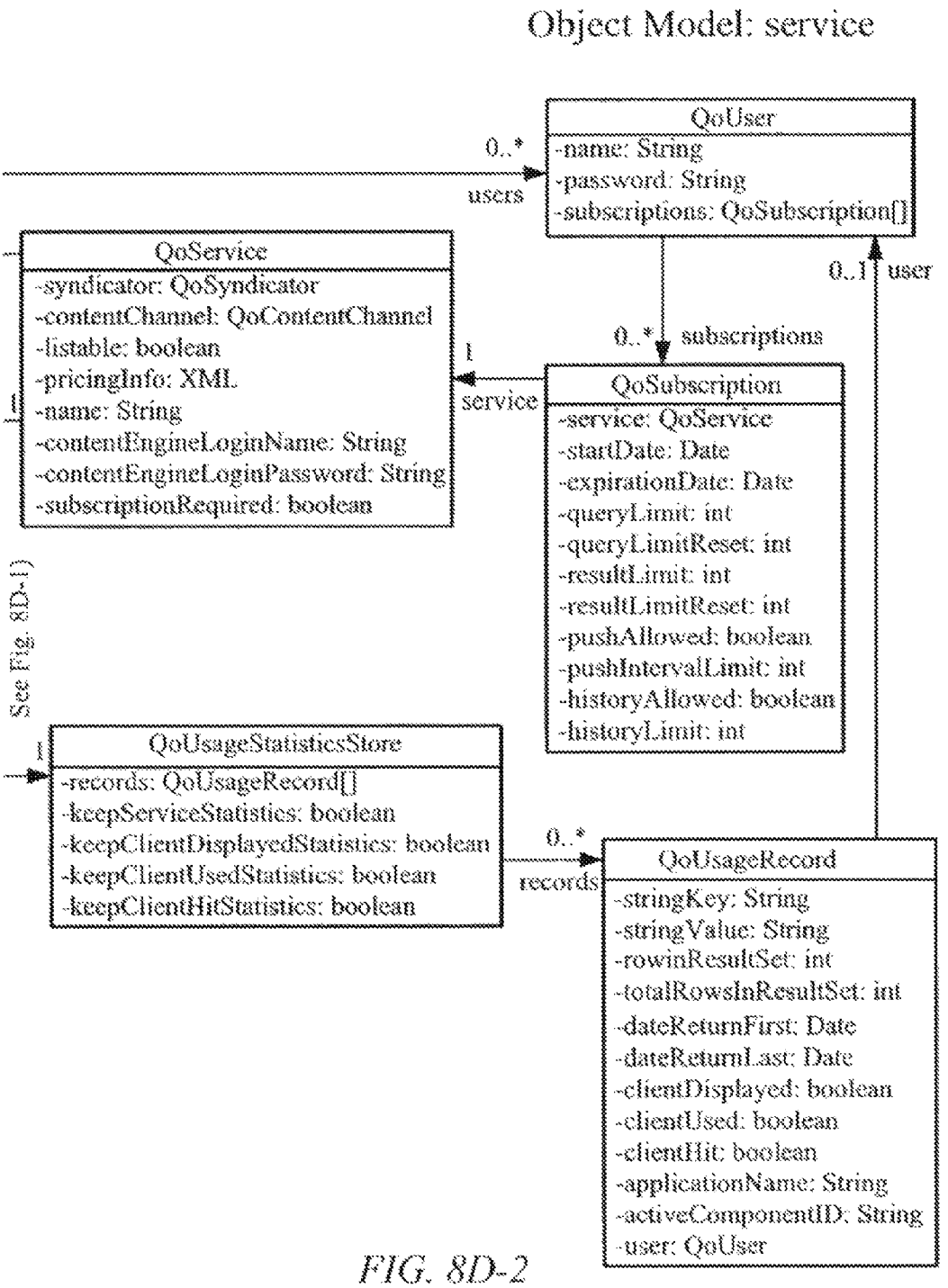

Object Model: service
QoUser
-name: String
-password: String
-subscriptions: QoSubscription[]
0..*
users
QoService
-syndicator: QoSyndicator
-contentChannel: QoContentChannel
-listable: boolean
-pricingInfo: XML
-name: String
-contentEngineLoginName: String
-contentEngineLoginPassword: String
-subscriptionRequired: boolean
1
1
0..*
subscriptions
1
service
QoSubscription
-service: QoService
-startDate: Date
-expirationDate: Date
-queryLimit: int
-queryLimitReset: int
-resultLimit: int
-resultLimitReset: int
-pushAllowed: boolean
-pushIntervalLimit: int
-historyAllowed: boolean
-historyLimit: int
0..1
user
See Fig. 8D-1)
1
QoUsageStatisticsStore
-records: QoUsageRecord[]
-keepServiceStatistics: boolean
-keepClientDisplayedStatistics: boolean
-keepClientUsedStatistics: boolean
-keepClientHitStatistics: boolean
0..*
records
QoUsageRecord
-stringKey: String
-stringValue: String
-rowinResultSet: int
-totalRowsInResultSet: int
-dateReturnFirst: Date
-dateReturnLast: Date
-clientDisplayed: boolean
-clientUsed: boolean
-clientHit: boolean
-applicationName: String
-activeComponentID: String
-user: QoUser

FIG. 8D-2

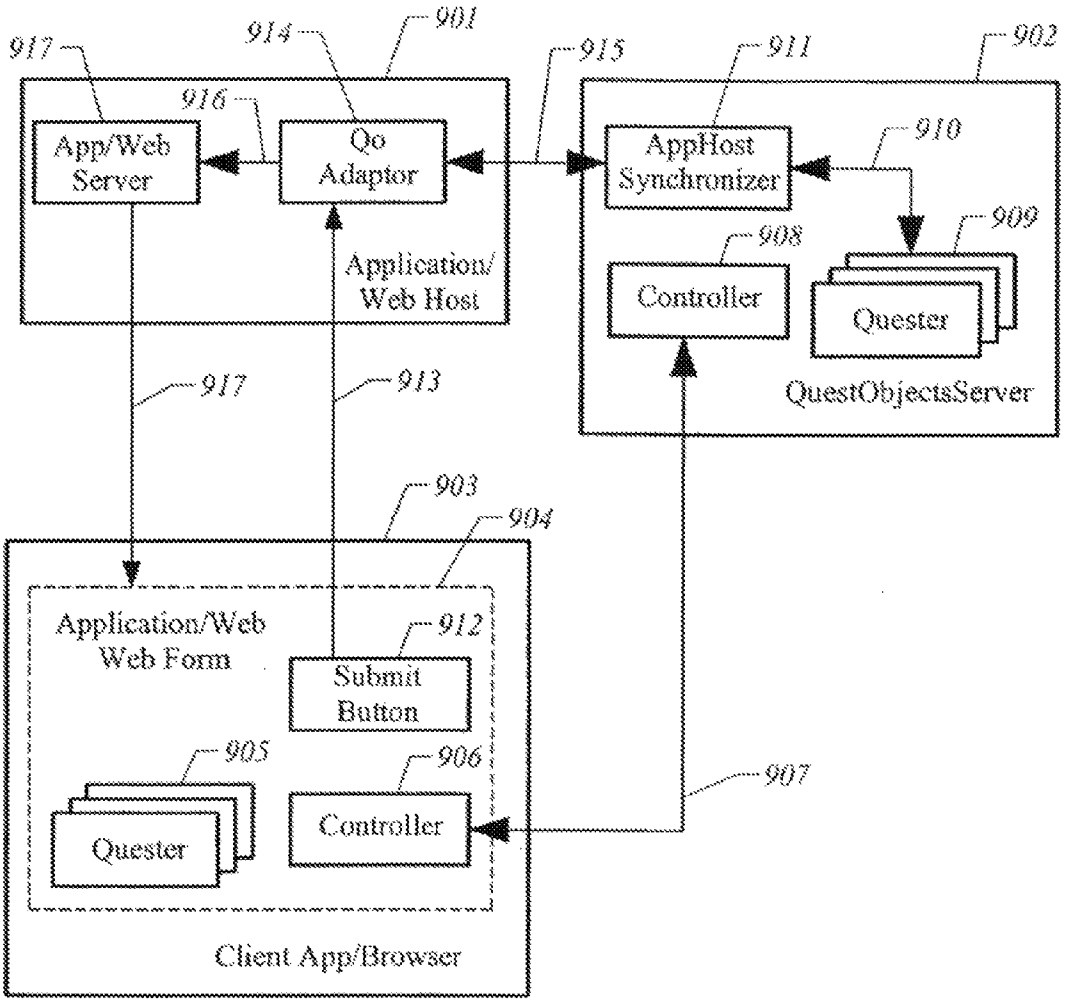

917
914
901
915
911
902
916
App/Web Server
Qo Adaptor
AppHost Synchronizer
910
908
909
Application/ Web Host
Controller
Quester
917
913
QuestObjectsServer
903
904
Application/Web Web Form
912
Submit Button
905
906
907
Quester
Controller
Client App/Browser

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. **3**A-**3**C 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. **3**A, 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. **3**B and **3**C 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. **3**A, 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. **3**B and example **313** shown in FIG. **3**C 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. **3**A. By clicking on the disclosure triangle **308** shown in the earlier FIG. **3**A (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. **3**B or a horizontal presentation mode **313**, shown in FIG. **3**C 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. **3**C 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. **3**B and **3**C, area **312** displays a QuestObjects "Thesaurus Service" (Thesa) that has been selected. Additionally, in FIG. **3**C 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. **8**A-**8**D illustrate and object model of an embodiment of the present invention. FIG. **8**A illustrates the base portion of the model containing the entities that are not specific to either QuestObjects Clients, QuestObjects Servers, or QuestObjects Services. FIG. **8**B displays the entities that are specific to the QuestObjects client. FIG. **8**C contains the entities specific to the QuestObjects Server. FIG. **8**D shows the entities specific to the QuestObjects Service.

Each of FIGS. **8**A through **8**D 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. **8**A is a reference to the 'QoService' entity in the figure of the service part, namely FIG. **8**D. 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. **8**A-*comprises* entities that are not specific to one of the tiers of the QuestObjects system. One of the most important entities shown in FIG. **8**A 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' prop-

erty 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. **8**B) and QoServerQuester (depicted in FIG. **8**C). 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. **8**A-**8**D 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. **8**A). 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 propery '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 AccessModule.

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 lan-

guages 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.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 8,539,024 B2
APPLICATION NO. : 13/366905
DATED : September 17, 2013
INVENTOR(S) : Smit et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

In the Claims

Column 33, line 34, Claim 20, delete "Previously Presented" and insert --new--.

Column 33, line 36, Claim 20, delete "Previously Presented" and insert --new--.

Column 33, line 37, Claim 20, delete "Previously Presented" and insert --new--.

Column 34, line 14, Claim 30, delete "Previously Presented" and insert --new--.

Signed and Sealed this
Eighth Day of April, 2014

Michelle K. Lee

Michelle K. Lee
Deputy Director of the United States Patent and Trademark Office