

**IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE**

INTERNATIONAL BUSINESS
MACHINES CORPORATION

Plaintiff

v.

ZYNGA INC.

Defendant.

§
§
§
§
§
§
§
§
§
§

C.A. No. 22-590-GBW

**HIGHLY CONFIDENTIAL –
ATTORNEYS’ EYES ONLY**

**Expert Report of
John K. Bennett, Ph.D., P.E.
February 2, 2024**

Table of Contents

1. Engagement	4
2. Qualifications	4
3. Materials Reviewed	7
4. Summary of Expected Testimony	7
5. Understanding of the Relevant Law	9
A. Claim Construction	9
B. Anticipation	10
C. Obviousness	11
D. Written Description	13
E. Level of Skill in the Art	13
F. Non-Infringing Alternatives	14
6. Summary of Relevant Technical Background	15
A. Distributed Systems and Applications	15
B. Object-Oriented Languages and Systems	16
C. Distributed Object-Oriented Systems	17
D. Smalltalk-80	18
E. Model-View-Controller (MVC)	19
F. The Internet, HTTP and TCP/IP Network Communication Protocols	22
G. JavaScript, Dynamic HTML (DHTML), and the Document Object Model (DOM)	28
1. JavaScript	28
2. DHTML and DOM	29
7. Overview of the '719 Patent and Asserted Claims	30
A. Overview	30
B. Asserted '719 Patent Claims	37
C. Infringement Considerations	40
8. Bases of Opinions	40
A. Background: the "Teachings" of the '719 Patent Were Not Novel to a POSITA	40
1. Techniques for Reducing Network Interaction Were Known and Practiced	40
2. The MVC Paradigm Was Known and Practiced	42
3. The Dual MVC Approach Was Known and Practiced	42
4. HTTP was Known, Practiced, and Built Upon Known Lower-Level Protocols	43
B. The Asserted Claims of the '719 Patent are Anticipated or Rendered Obvious by the Prior Art	44

1.	Lowery Anticipates or Renders Obvious Asserted Claims 1, 2, 12, and 26 of the '719 Patent	45
2.	Rover Anticipates or Renders Obvious Asserted Claims 1, 2, 12, and 26 of the '719 Patent	86
3.	Hyndman Anticipates or Renders Obvious Asserted Claims 1, 2, 12, and 26 of the '719 Patent	130
4.	Bennett Anticipates or Renders Obvious Asserted Claims 1, 2, 12, and 26 of the '719 Patent	175
5.	Yelon Anticipates or Renders Obvious Asserted Claims 1, 2, 12, and 26 of the '719 Patent	207
6.	Roy Anticipates or Renders Obvious Asserted Claims 1, 2, 12, and 26 of the '719 Patent	244
7.	Additional Opinions Relating to Dependent Claims 2 and 12	296
8.	No Secondary Considerations Render the Asserted Claims Nonobvious	299
9.	The Asserted Claims May Lack Written Description, Depending on Plaintiff's Infringement Argument	300
10.	Non-Infringing Alternatives	303
A.	Non-Infringing Alternative No. 1	303
B.	Non-Infringing Alternative No. 2	305
11.	Concluding Remarks	307



1. Engagement

1. In September 2023, I was retained by counsel to Zynga to evaluate certain aspects of U.S. Patent 7,702,719 B1 (the '719 Patent), and to offer my own analysis and opinions regarding matters related to that Patent. I understand that Plaintiffs are asserting only Claims 1, 2, 12, and 26 of the '719 Patent (the Asserted Claims); I have considered only those claims in the preparation of this report. In my reading and understanding of the '719 Patent, I have been guided by the Court's Construction of Claim Terms for that Patent.¹

2. I am prepared to be called as an expert witness if my testimony is requested in this matter. This report contains a summary of my analysis to date. I am continuing my work on the case, and I may supplement, refine, or revise my opinions and findings as a result of further review and analysis. I am being paid \$800 per hour for work in this matter. I have no business relationships with Zynga or IBM.

2. Qualifications

3. I am a Professor Emeritus of Computer Science and Electrical and Computer Engineering at the University of Colorado at Boulder. Over my career, my research has focused broadly in the area of operating system/microprocessor architecture interaction and, more narrowly, in the area of distributed and parallel computing systems. My teaching has included graduate and undergraduate courses in computer architecture, operating system design, distributed systems, computer system design, and other topics.

4. My titles at the University of Colorado included Director of Inworks, Associate Vice Chancellor for Innovation Initiatives, and Professor of Computer Science at the University of Colorado Denver | Anschutz Medical Campus; and Archuleta Professor of Computer Science

¹ *IBM v. Zynga Inc. et al.*, No. 1:22-cv-00590-GBW (D. Del. October 30, 2023), ECF No. 171.



and Engineering and Professor of Electrical, Computer and Energy Engineering at the University of Colorado at Boulder. I served as the Director of the ATLAS Institute and Associate Dean of Engineering for Education in the College of Engineering and Applied Sciences at the University of Colorado at Boulder, and as the Master of Richardson College at Rice University.

5. I have a doctorate in Computer Science from the University of Washington. My doctoral dissertation involved the design, implementation and evaluation of a Smalltalk system distributed over a local area network. Smalltalk is an object-oriented programming environment developed at the Xerox Palo Alto Research Center that was a precursor of Oracle's Java technology. I have done pioneering work in both distributed shared memory systems and adaptive software techniques for shared memory parallel computing. Prior to completing my thesis work at the University of Washington, I was President of Pacific Mountain Research, Inc. (PMR) in Seattle, Washington.

6. For the past forty-five years, I have focused on the development of computer software and hardware, both directly as a creator of sophisticated software applications and operating systems, and as an academic, studying and contributing to the field. Examples of commercial and academic computer systems that I have helped develop include the mission control system for Jet Propulsion Laboratories, the control system for the Washington D.C. Metropolitan Transit Authority light rail system, the propulsion control system for the Spruance class gas-turbine-powered destroyer, the Eden distributed computer system, an object-oriented distributed computing system based upon Smalltalk, several multiprocessor and virtual memory systems, several high performance graphics and image processing systems, and several very large scale integrated (VLSI) circuits, biomedical instrumentation systems, and video processing systems. These systems were utilized or marketed by several companies, including Motorola, Signetics,

[REDACTED]

RCA, IBM, Eastman Kodak, Microsoft, Sequent, Texas Instruments, Weyerhaeuser, the Department of Defense, and the New York Stock Exchange.

7. I have published over eighty research papers, and have given presentations at numerous conferences, universities, and research laboratories. A list of these papers can be found in Appendix A.

8. Over the years, I have received significant research funding from industry and government organizations in support of my work in over thirty-one research and educational projects, including support from the National Science Foundation, the Defense Advanced Research Project Agency, NASA, the National Institute of Health, the National Institute of Disability and Rehabilitation Research, and the Texas Higher Education Coordinating Board, as well as grants from Compaq, Intel, Tandem, Microsoft, Ansell, Regent, Bell Northern Research, Hewlett-Packard, and IBM. (A listing of my grants for the past fifteen years is attached as part of Appendix A.)

9. I have been an active member of the Association for Computing Machinery since 1979, and the Institute for Electrical and Electronic Engineers (in which I hold the rank of Senior Member) since 1976. I am a registered Professional Engineer (inactive) in the State of Texas. I am the past president of the national Governing Board of Engineers Without Borders USA, and have received several teaching awards, including the W.M. Keck Foundation National Award for Engineering Teaching Excellence. I am a named inventor on two U.S. Patents.

10. As a result of this experience, I am thoroughly familiar with distributed and object-oriented system design and development, hardware and software development, and software testing and support. I have participated in the development and testing of commercial software products as both a software developer and manager. In the course of my research and teaching, I

[REDACTED]

have also become broadly familiar with various client and server operating systems, and a variety of applications development and enterprise programming environments.

11. Over the past thirty plus years, I have been qualified as an expert in computer systems in courts throughout the United States and Canada, and have given testimony on subjects relating to computer software in more than a dozen cases. My expert testimony has included analysis of computer software behavior, comparisons of functionally similar software and hardware, patent claim interpretation and patent claim analysis. A listing of the cases in which I have testified is contained in Appendix B.

3. Materials Reviewed

12. My knowledge in these areas (and, therefore, my opinions) derives in part from my active involvement in computer systems research and development throughout my career. As part of my teaching and academic advancement, I have kept current on issues in computer software, including (specifically) operating and distributed system development.

13. Aside from my background in computer science, in the course of preparing this report I have taken into consideration the following materials:

- a) the various complaints, motions, briefs, interrogatory responses and other pleadings submitted by the parties in this matter;
- b) publicly available documents relating to issues in the case, including patents, articles, books, and websites; and
- c) other documents and information specifically cited in this report.

4. Summary of Expected Testimony

14. I generally expect to testify on the following subjects:

- a. The level of knowledge that would be expected of a person of ordinary skill in the art (POSITA) during the time frame relevant to the '719 Patent.



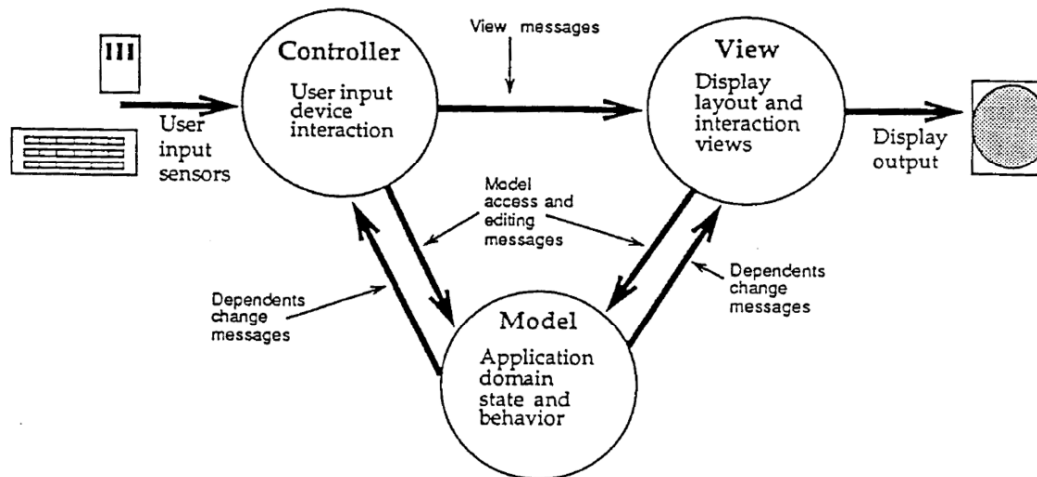
Second, the reactive principle requires that *every component accessible to the user should be able to present itself in a meaningful way for observation and manipulation.*

61. Of particular relevance to this case, these two principles led to the Model-View-Controller (MVC) – based graphical user interface design that was an essential component of Smalltalk-80.

E. Model-View-Controller (MVC)

62. Model-View-Controller (MVC) is an object-oriented user interface design methodology, or *metaphor*, that divides the user interface into three related components: Model, View, and Controller. It was created during the development of Smalltalk-80. As implemented in Smalltalk-80, objects of three classes made up the MVC: the *Model*, which maintained the state of an object, the *View*, which creates a representation of that state for viewing, and the *Controller*, which managed user interaction between the Model and the View (i.e., updates to the object's state). A Model can be simple, e.g., an integer counter, or complex, e.g., a database. Controllers communicate with the Model, and provide the interface between the Model, its associated Views (of which there may be several), and the user interface. Applications in Smalltalk-80 usually, but not always, had a single Model, and several view/controller pairs. Views request data from their model, and create representations of objects' data that they receive for subsequent display. These relationships are shown in the figure below.¹²

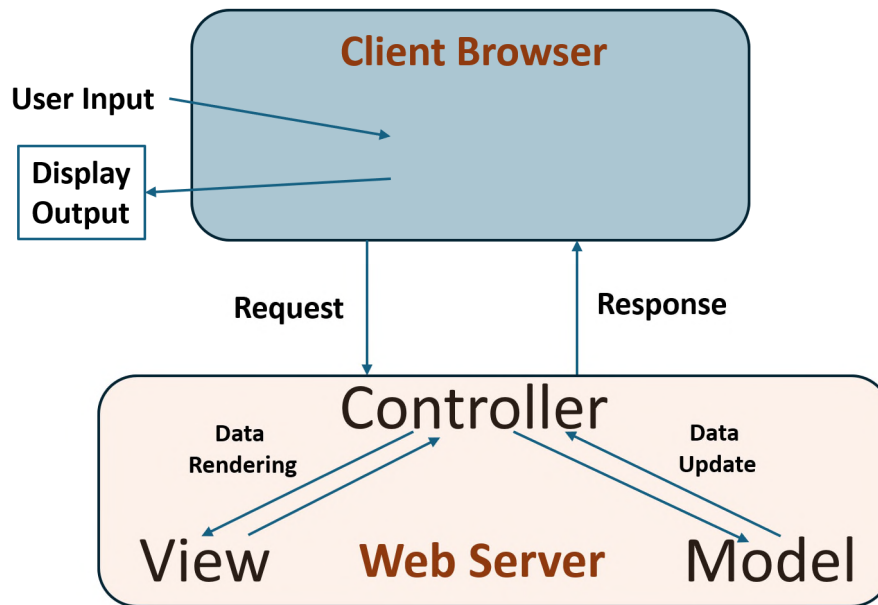
¹² G. Krasner & S. Pope, *A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System*, Journal of Object-Oriented Programming, Vol. 1, p. 5, Jan. 1988.



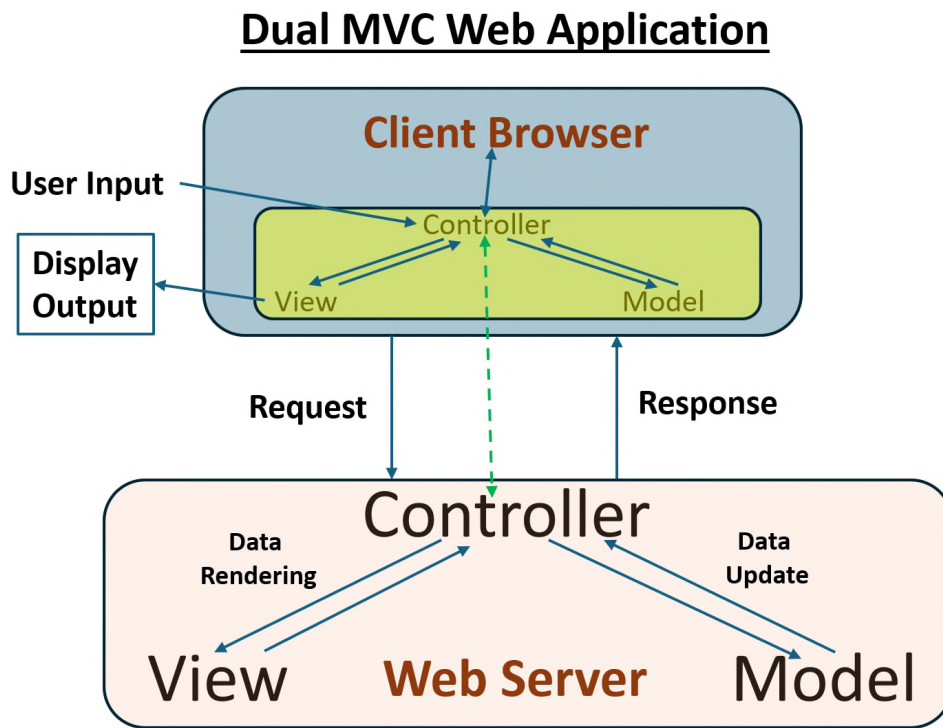
63. Controllers typically maintain the interface between their associated Models and Views and the user interface devices (keyboard, pointing device, etc.). Put simply, a Model is a collection of data (e.g., a list of library books); a View creates representations of this data (e.g., sorting the books by author, title, or subject); and a Controller provides mechanisms for manipulating the model (e.g., adding or checking out a book). In this approach, Views and Controllers have exactly one Model, but a Model can be associated with several Views and Controllers. As shown in the figure above, Smalltalk-80 display output is managed by a View. Other MVC systems place this responsibility with a Controller.

64. Although MVC was developed for desktop graphical user interface, its simplicity, encapsulation, and modularity made it popular for designing early web applications. This usage increased after NeXT incorporated MVC into WebObjects (first released in 1996). When using MVC for web application design, the client browser typically becomes another just user interface mechanism (and a gateway for other user interactions), as depicted in the figure below.

Standard MVC Web Application



65. As mechanisms for embedding code within MVC-based browsers (e.g., JavaScript) were developed in the 1990s, it became natural to begin dividing MVC responsibilities between server and client. Although dual MVC distributed applications had been built prior to MVC use in web application development, e.g., Distributed Smalltalk, dual MVC became more widespread with the advent of web application software environments that facilitated such development. The figure below depicts a common configuration of the dual MVC model employed in a web application. In this example, portions of the application Model, View and Controller reside on the client. Typically, these portions are cached copies of some part of their associated component located on the server. Web applications communicate using HTTP at the application level. The dashed green line depicts “private” Controller to Controller communication that may be embedded into the HTTP communication.



66. MVC evolved over time, giving rise to several variants, including Hierarchical Model–View–Controller (HMVC), Presentation–Abstraction–Control (PAC), Model–View–Adapter (MVA), Model–View–Presenter (MVP), Model–View–View-Model (MVVM), and others.

F. The Internet, HTTP and TCP/IP Network Communication Protocols

67. The Internet has a storied history dating back to the early 1960s. The technologies that underly the Internet: a network of networks of computers, packet switching, and hypertext had their roots in the groundbreaking work of Kleinrock (packet switching on digital networks), Licklider (“Intergalactic Computer Networks”), Baran (store and forward message routing), and Nelson and van Dam (hypertext).

68. The communication protocols used to send and receive network messages became standardized when Cerf and Kahn published “A Protocol for Packet Network



82. DHTML and the DOM allowed website developers to add effects that were otherwise difficult to achieve, simply by changing the Document Object Model (DOM) and page style. This combination of HTML, CSS, and JavaScript offered ways to:

- animate text and images in web documents,
- embed dynamic content (e.g., a counter) that automatically refreshed its content without communication with the server,
- capture user input (such as pressing a key and clicking the mouse), and then process, verify and respond to that input without having to send data back to the server, and
- incorporate rollover buttons and drop-down menus into web pages.

83. These website development capabilities, which were routinely available at the time the '719 Patent was filed, appear to encompass the claimed inventions of the '719 Patent.

7. Overview of the '719 Patent and Asserted Claims

A. Overview

84. The '719 Patent describes a dual MVC-based web application intended to be employed in a client/server distributed system. In particular, the '719 Patent seeks to both improve application performance and ease software development and maintenance efforts:

“The invention relates to web applications and, more particularly, to an architecture and method of programming which improves performance and eases the development and maintenance effort. The invention addresses performance by employing a dual-MVC approach, in which a subset of the application’s Model-View-Controller reside on the client, and the full Model-View-Controller and View-Generating Logic reside on the server, thereby reducing the number of required server interactions. In contrast to the fat-client architecture, the invention does not require any application installation or persistence on the client.”²⁴

²⁴ '719 Patent; 2:39-49.



85. The described performance improvement is attributed in part to the application downloading small portions of code that contain MVC software. This software is downloaded when needed (“upon demand”) and executed without the need to permanently install the software:

“The dual-MVC approach of the invention avoids the installation and disk-space problems because client-side code associated with the MVC of the application is preferably downloaded from the server on demand by the user of the client device, avoiding the install process, and is preferably not saved permanently to the disk of the client.”²⁵

86. Thus, at various times during the execution of the application, portions of the application’s model, view, and controller reside on the client. The server always contains the authoritative version of the application’s model, view and controller:

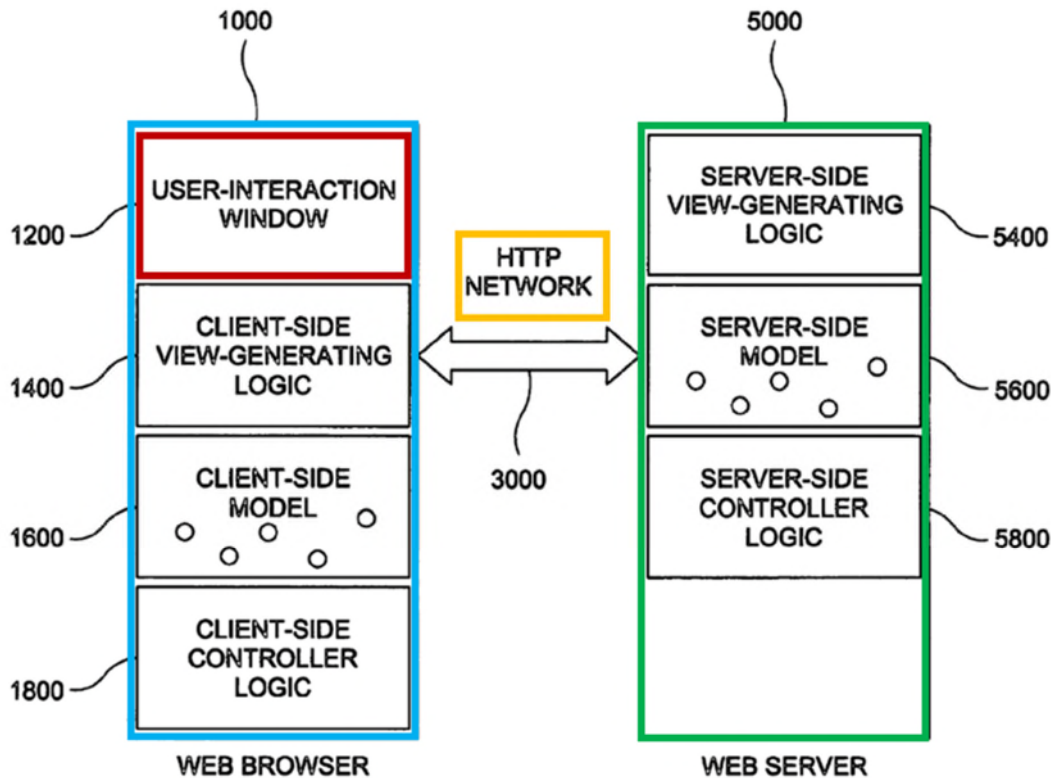
“The invention addresses performance by employing a dual-MVC approach, in which a subset of the application’s Model-View-Controller reside on the client, and the full Model-View-Controller and View-Generating Logic reside on the server, thereby reducing the number of required server interactions.”²⁶

87. This idea of the model, view, and controller being located on both the client and the server is depicted in the block diagram of the “dual-MVC (Model-View-Controller) approach” shown below.²⁷

²⁵ ’719 Patent; 3:1-6.

²⁶ ’719 Patent; 2:42-47.

²⁷ ’719 Patent; Figure 3 (coloration added).



88. In this figure, the “User Interaction Window” 1200 is a web browser. All other boxes are the individual components of the Model-View-Controller found on both client (surrounded by a blue box) and server (surrounded by a green box). The “HTTP Network” 3000 is presumably a standard network interface capable of supporting HTTP, although the version of HTTP to which the patent’s authors refer is not specified. The server-side MVC components (5000) are considered to be the “authoritative” (complete) copies of these components; the client-side versions of these components (1000) are subsets of the server version (“a subset of the application’s Model-View-Controller reside on the client, and the full Model-View-Controller and View-Generating Logic reside on the server”²⁸).

²⁸ ’719 Patent; 2:43-46.

6. Roy⁶¹⁸ Anticipates or Renders Obvious Asserted Claims 1, 2, 12, and 26 of the '719 Patent

477. U.S. Patent 5,838,909 (Roy) is entitled “Reducing Latency When Synchronizing Access to a Multi-User Database Over a Network.”⁶¹⁹ Roy was filed May 23, 1996, and issued Nov. 17, 1998. Because its filing date precedes that of the '719 Patent, I understand that Roy meets the legal standards required to qualify as prior art to the '719 Patent. Roy is neither identified on the face of the '719 Patent, nor was it considered during prosecution.

478. In my opinion, Roy anticipates or renders obvious all the limitations of Asserted Claims 1, 2, 12, and 26 of the '719 Patent. In the alternative, to the extent any limitation is found missing from Roy, Roy renders all the Asserted Claims of the '719 Patent obvious in light of the knowledge possessed by one of ordinary skill in the art and/or the specific prior art references I discuss below. The bases for this opinion are set forth below.

479. Note: In some instances, Roy uses the terms “Master” and “Slave” to refer to components of the described invention. These terms, which trivialize the horrific practice of treating people as property, are not part of the modern engineering lexicon. Since these terms are essential to being able to describe the patent in the language in which it was written, I will, where necessary, also use them here.

a. A Brief Description of U.S. Patent 5,838,909 (Roy)

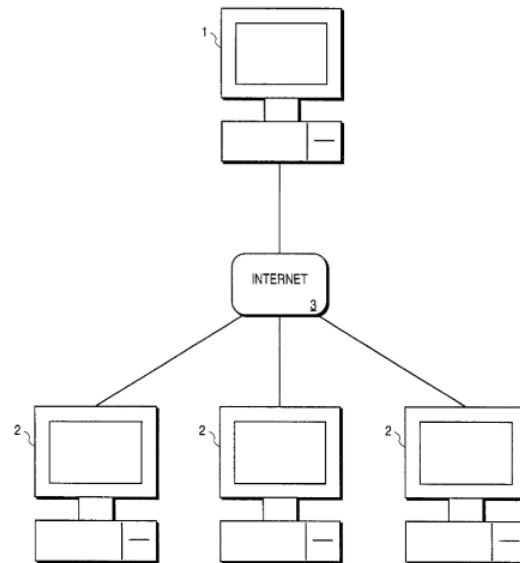
480. Roy generally discloses a distributed software system targeted toward multi-player games on the Internet. As shown in the figure below, Roy discloses a “central server computer 1” coupled via the “Internet 3” to a number of “client computer systems 2.”⁶²⁰ As disclosed by Roy,

⁶¹⁸ U.S. Patent 5,838,909 (“Roy”).

⁶¹⁹ Roy; Cover.

⁶²⁰ Roy; 3:48-51 and Figure 1.

both the server and all clients execute the game software referred to in Roy as an “event server,” as further described below.



481. Each node (i.e., computer or server) in the system contains a *database* and an *event server*. The database at each node contains a model of a fictitious “game world.” The event server at each node is responsible for processing local user inputs (“events”) that update the game world database, as well as synchronizing with other nodes’ event servers (via the “master event server” described below), so that their copies of the database are kept up to date.⁶²¹

482. As disclosed in Roy, one database is designated as the “master database,” and one event server is designated as the “master event server.” These components are typically located on the same node. The “master database” is considered to be the authoritative copy of game state. Roy teaches that the node so designated should be the node whose user is most active (i.e., the one with the most restrictive latency requirements). The idea is that by placing the “master” components near the most active user, in-game latency will be reduced.⁶²²

⁶²¹ Roy; 1:66-2:10.

⁶²² Roy; 3:16-47.

483. Since the most active user is likely to change during game play, Roy also teaches a technique for migrating the master event server to another node (the one whose user has become most active). The figure below depicts these relationships.⁶²³

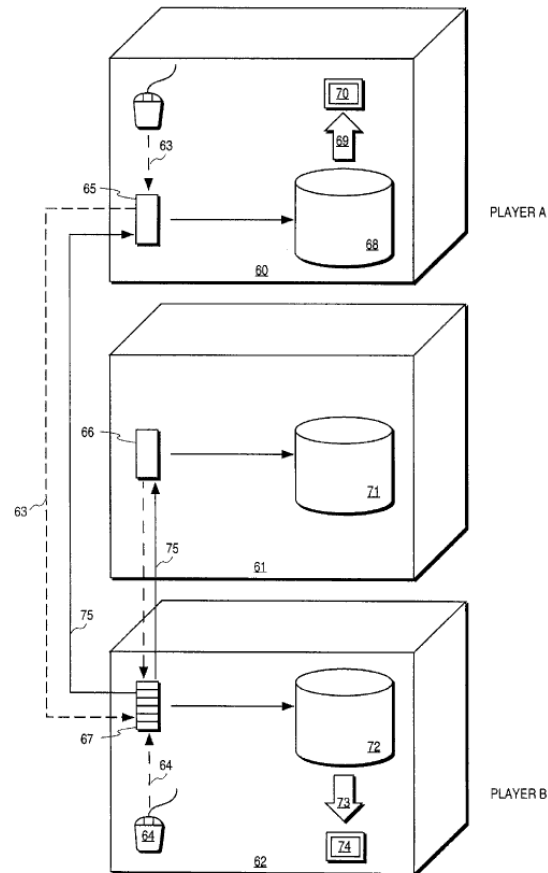


FIG. 6

484. Slave copies of the master database (or portions thereof) are stored on each machine other than the machine in which the master event server is located. By providing one master copy of the database, events originating from different machines can be more easily synchronized. By providing slave copies of the database in every other machine involved in the game, the need to send rendering commands across the network is reduced or eliminated, which reduces bandwidth consumption.⁶²⁴ An event server generally performs four primary functions:

⁶²³ Roy; Figure 6 and 8:60-9:5.

⁶²⁴ Roy; 4:47-54.

1. connecting to other event servers,
2. routing events to the local database and to other event servers (including the master event server),
3. migrating the master event server when appropriate, and
4. reconnecting to a new event server such that game events are not lost or duplicated.⁶²⁵

485. The figure below depicts an implementation of a multi-user game played over the Internet as disclosed by Roy.⁶²⁶

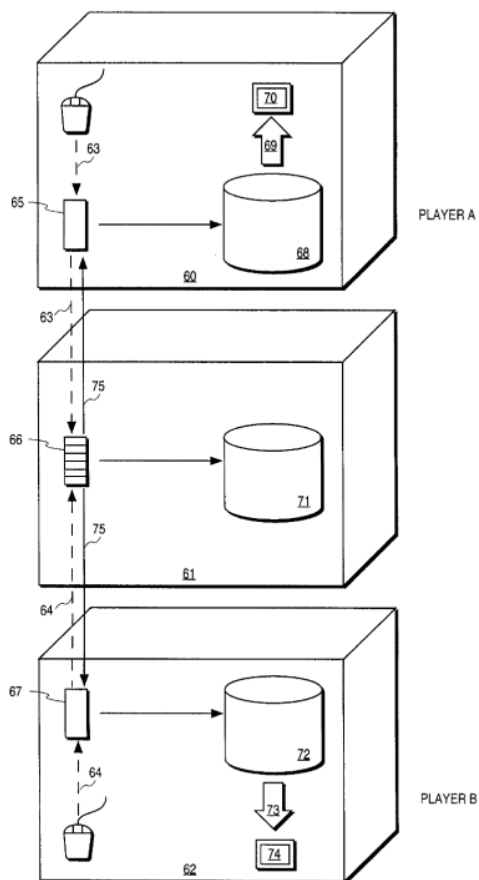


FIG. 5A

486. This figure depicts a game played by two players, PLAYER A and PLAYER B. PLAYER A is the user operating computer system 60; PLAYER B is the user operating computer

⁶²⁵ Roy; 4:62-5:2.

⁶²⁶ Roy; Figure 5A.



system 62. Initially, the master database 71 and the master event server 66 are located in a central server computer system 61. Also shown within computer system 60 (used by Player A) are a slave event queue 65 and a slave copy 68 of master database 71. Similarly, located in computer system 62 (used by player B) is a slave event server 67 and a slave copy 72 of master database 71.⁶²⁷

487. The display 74 shown within computer system 62 is updated by rendering commands 73 from the slave copy 72 of master database 71, rather than from the master database 71. This design choice is intended to reduce bandwidth consumption on the network.⁶²⁸

488. In the same manner, the display 70 shown within computer system 60 is updated by rendering commands 69 from slave copy 68 of master database 71, rather than directly from master database 71.⁶²⁹

489. User events 64 (such as keyboard presses or mouse movement or button clicks) from computer system 62 are received by slave event server 67 and then passed along to the master event server 66, unless these events are purely local (events that are only local to computer system 62). In the same manner, events 63 originating from computer system 60 are received first by slave event server 65 and then passed along to master event server 66, unless those events are purely local to computer system 60. Events received by the master event server 66 are used to update the master database 71. These events are also passed along to all slave event servers as master events 75.⁶³⁰

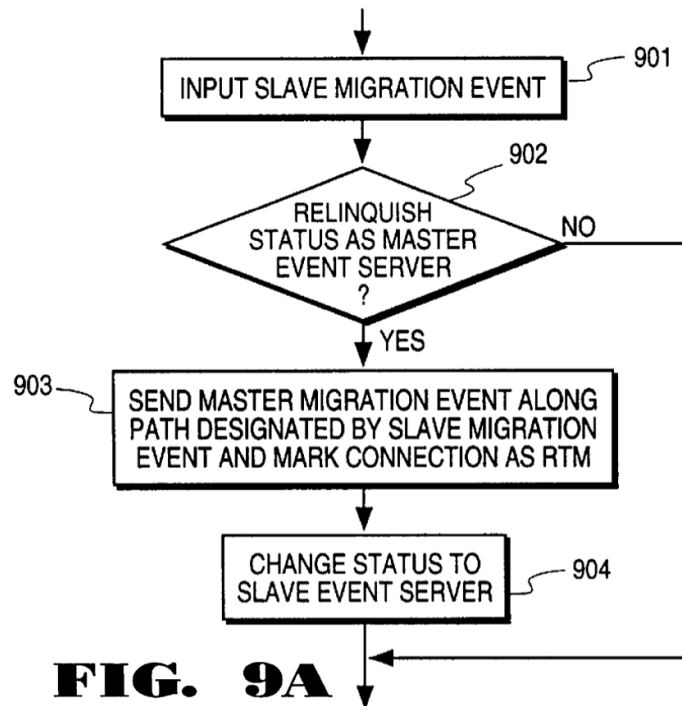
⁶²⁷ Roy; Figure 5A and 5:17-29.

⁶²⁸ Roy; Figure 5A and 5:29-33.

⁶²⁹ Roy; Figure 5A and 5:33-36.

⁶³⁰ Roy; Figure 5A and 5:37-49.

490. When a determination is made of which computer system participating in the game is used by the player having the role that requires the least latency, the master event server 66 is migrated to the machine used by that player, as depicted in the figure below.⁶³¹



491. This figure depicts the process of migration from the viewpoint of the master event server. The process begins when the current master event server receives a request in the form of a slave migration event (step 901). This request would typically come from a client that has determined that its user requires lower latency (i.e., this user is very active in game). The master event server then makes a determination (using game logic criteria not relevant to this discussion) whether to relinquish its status as the master event server (step 902). If the decision is made to relinquish its status as the master event server, then the master event server sends a “master migration event” along the path by which it received the slave migration event, which will ultimately reach the requesting client. All computers along that path will mark the network path to

⁶³¹ Roy; Figure 9A and 5:65-6:2.

the requesting client as the Route to Master (RTM) (step 903). The master event server will then change its status to slave event server (step 904).⁶³²

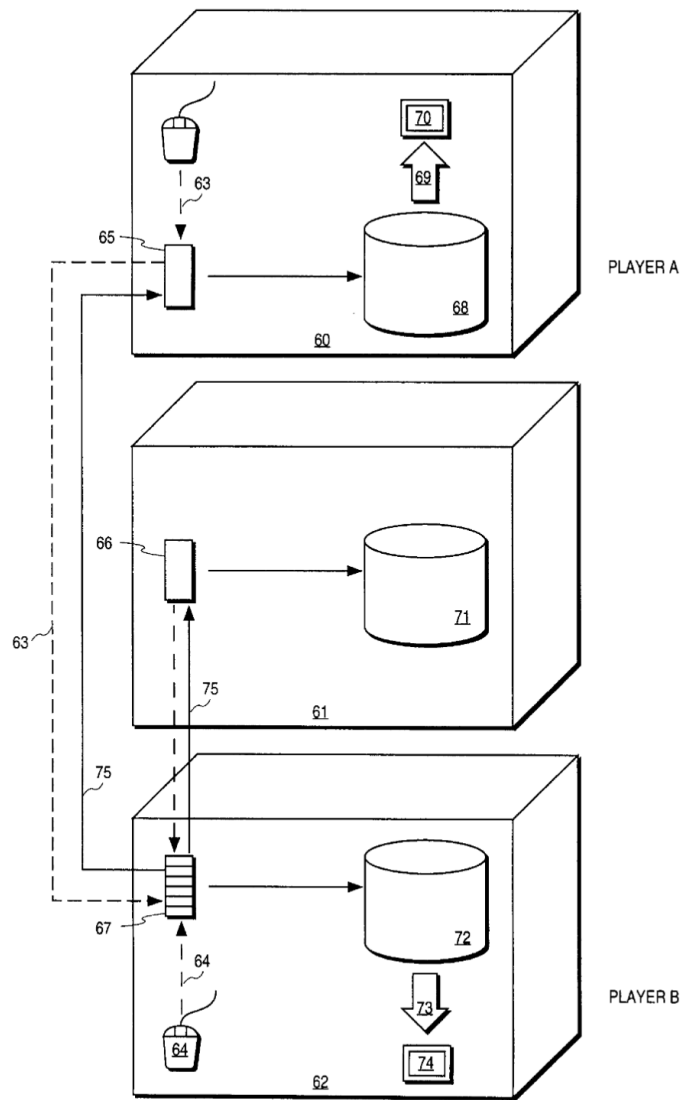


FIG. 6

492. More internal details can be seen in the figure above, where each box (60, 61, and 62) represent nodes. The cylinders 68, 71, and 72 represent copies of the local database within these nodes that contain the game state. Boxes 70 and 74 represent the displays on these nodes.

⁶³² Roy; Figure 9A and 7:60-8:5.



Three-button mice (63 and 64) generate user input events, which are handled by event servers 65 and 67, respectively. Event server 67 (with small horizontal lines) is the “master event server”.⁶³³

493. Thus, we can relate the language of Roy to that of the ’719 Patent, as follows: the Roy “master database” and “master event server” constitute the “server” as described in the ’719 Patent; and the Roy “slave database” and “slave event server” constitute the “client” as described in the ’719 Patent.

494. Roy also teaches an embodiment in which the master database is divided into multiple sections, or “objects”, each of which may reside on a different machine. Each of these disparate sections (located on different nodes) functions as a “master database” relating to a particular aspect of the game. A separate “master event server” is associated with each of these sections.⁶³⁴

b. Roy Anticipates or Renders Obvious all Elements of Claim 1 of the ’719 Patent

[1P] A method for use in a client/server system of reducing interactions between a client and server in association with an application being accessed by the client at the server, the method comprising the steps of:

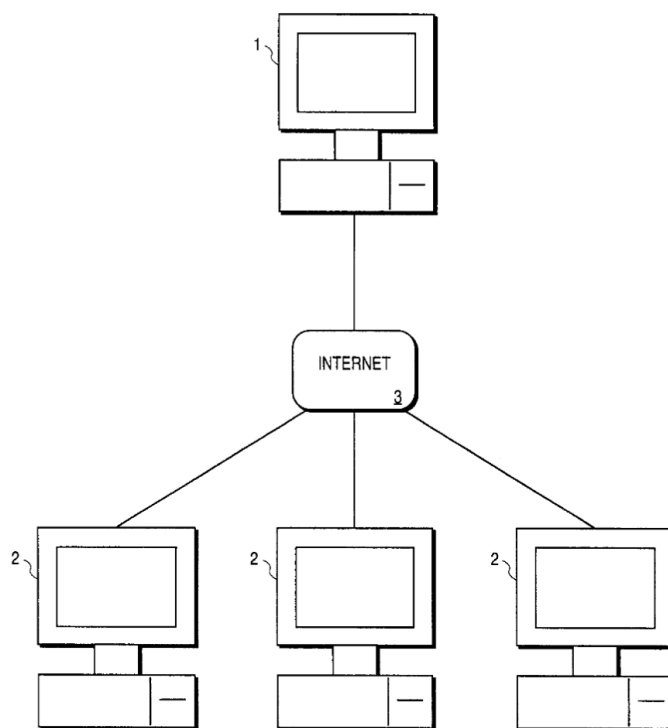
495. To the extent the preamble is limiting, in my opinion, Roy discloses a method for use in a client/server system (e.g., the client/server systems depicted in Figures 1, 4, 5, and 6) of reducing interactions between a client and server (e.g., Roy discloses that interactions between client computer 60 and server computer 61 when client computer 60 processes a “local event” do not require any interaction with the central server 61) in association with an application being accessed by the client at the server (e.g., a user connects to an online multiplayer game).

⁶³³ Roy; Figure 6 and 8:60-9:5.

⁶³⁴ Roy; 4:56-61.

496. First, Roy discloses a client/server system. Roy explains the operation of the claimed system as follows:

“Referring now to FIG. 1, a network configuration in which the present invention may be implemented is illustrated. A central server computer 1 is coupled via the Internet 3 to a number of client computer systems 2. The server computer system 1 and each of the client computer systems 2 includes stored game software, including software in which the present invention is embodied. The game software includes various application program interfaces (APIs) providing an interface to operating system software.”⁶³⁵



“A method is provided of reducing network latency in accessing a database in a network. The network has a number of nodes, including a first node and a second node. The first node includes a first event server for sequencing commands to the database, which is stored in one of the nodes in the network. The second node includes a second event server for sequencing commands to the database. The method comprises the steps of changing the status of the first event server from master event server to slave event server based on a network latency criterion, and assigning the status of master event server to the second event server based on the network latency criterion.”⁶³⁶

“FIG. 4B illustrates a multi-player game running in a network using a client-server configuration.”⁶³⁷

⁶³⁵ Roy; Figure 1 and 3:48-56 (highlighting added).

⁶³⁶ Roy; 1:66-2:10 (highlighting added).

⁶³⁷ Roy; 2:31-32 (highlighting added).



“FIG. 4B shows a well-known configuration for implementing a multi-user game on a network. In the configuration of FIG. 4B, the game world is provided by a database 53 located on a central server 51. Client computer systems 50 and 52 are used by Players A and B of the game, respectively. Software 54 located in the central server 51 receives events 58 from computers 50 and 52 and provides these events to the database 53 in order to update the game world. Rendering commands 56 are then sent from the updated database 53 in the central server 51 across the network to update the displays 55 and 57 of client computers 50 and 52, respectively. As mentioned above, implementing as shown in FIG. 4B on the Internet can be problematic if either of players A and B are subject to a significant latency restriction. In addition, the transmission of rendering commands 56 from server 51 to clients 50 and 52 can be hindered by bandwidth limitations of the network. Consequently, the present invention provides a technique for overcoming such difficulties. In the technique of the present invention, every machine that participates in the game runs software referred to herein as an ‘event server.’ One of the event servers is known as the ‘master event server’ and is responsible for sequencing all changes to a master database representing the game world. All event servers other than the master event server are considered to be ‘slave event servers’. In the preferred embodiment, the master database is located on the same machine as the master event server. Slave copies of the master database (or portions thereof) are stored on each machine other than the machine in which the master event server is located. By providing one master copy of the database, events originating from different machines can be more easily synchronized. By providing slave copies of the database in every other machine involved in the game, the need to send rendering commands across the network is reduced or eliminated, which reduces bandwidth consumption.”⁶³⁸

497. Second, Roy discloses a method for reducing client/server interaction in association with an application being accessed by the client. For example, Roy explains how the need to send rendering commands across the network is reduced by providing slave copies of the database to every computer connected to the game, and how, for example, local events are processed locally (i.e., without server interaction) by a respective gamer’s computer, and the results are then displayed on that player’s computer:

“By providing one master copy of the database, events originating from different machines can be more easily synchronized. By providing slave copies of the database in every other machine involved in the game, the need to send rendering commands across the network is reduced or eliminated, which reduces bandwidth consumption.”⁶³⁹

⁶³⁸ Roy; 4:18-54 (highlighting added).

⁶³⁹ Roy; 4:48-54 (highlighting added).

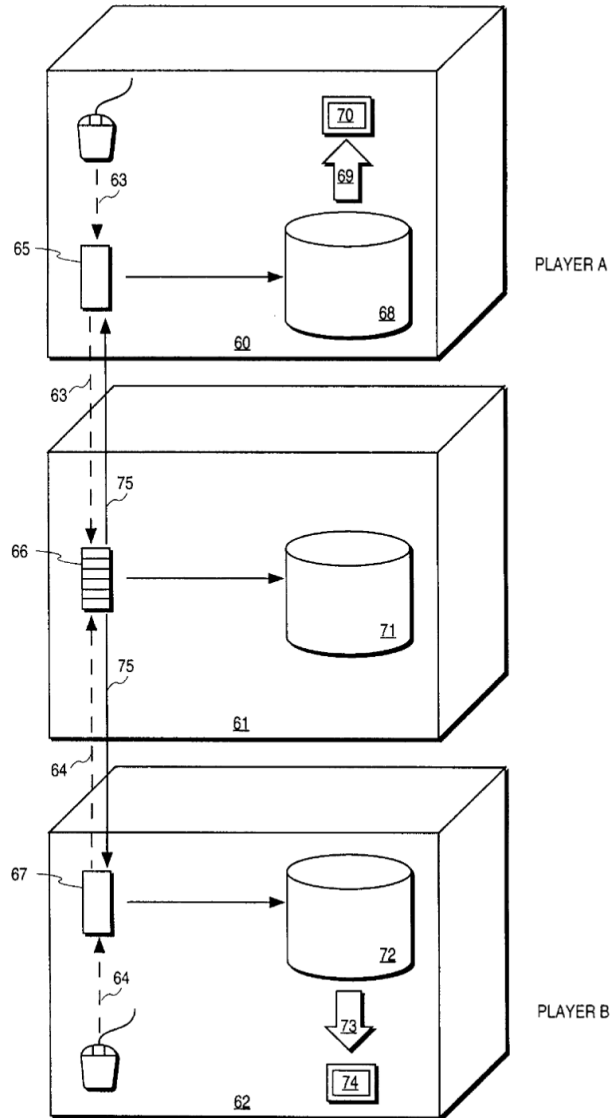


FIG. 5A

“FIG. 5A illustrates a technique according to the present invention for implementing a multi-user game over the Internet. In FIG. 5A, the game is played by two players, players A and B, operating computer systems 60 and 62, respectively. It is noted, however, that the present invention is equally applicable to a game played simultaneously by more than two players. Initially, the master database 71 and the master event server 66 are located in a central server computer system 61. Located in computer system 60 (used by player A) are a slave event queue 65 and a slave copy 68 of master database 71. Similarly, located in computer system 62 (used by player B) is a slave event server 67 and a slave copy 72 of master database 71. The display 74 of computer system 62 is updated by rendering commands 73 from the slave copy 72 of master database 71, rather than from the master database 71 itself, thereby reducing bandwidth consumption on the network. Similarly, the display 70 associated with computer system 60 is updated by rendering commands 69 from slave copy 68 of master database 71, rather than directly from master database 71. Events 64 from computer system 62 are received

by slave event server 67 and then passed along to the master event server 66, unless the events 64 are purely local events for updating only the slave copy 72 of the database. Purely local events might be, for example, events which set display options or other functions that are only relevant to computer system 62. Similarly, events 63 originating from computer system 60 are received first by slave event server 65 and then passed along to master event server 66, unless those events are purely local to computer system 60. Events received by the master event server 66 are used to update the master database 71 and are passed along to all slave event servers as master events 75.”⁶⁴⁰

498. For example, as described, “local events” are processed on the client and therefore do not require any server interaction.⁶⁴¹ Accordingly, interactions between the client and server are reduced. In particular, Roy discloses that an “event” is created each time a user issues a command. If the event is determined to be a local event, then it is processed by the client and the results displayed.⁶⁴² As noted above, examples of local events include “events which set display options or other functions that are only relevant to [that] computer system.”⁶⁴³

499. As another example, Roy teaches that, if a player’s computer becomes an effective server (i.e., the master event server and master database are migrated to that computer), then all events generated by that player are processed locally, regardless of whether they are local events or master events (i.e., interactions are reduced with the other computers including central server 61).⁶⁴⁴ Thus, for example, if the master event server and master database were migrated from central server 61 to computer system 62, all events generated by player B on computer system 62 would be processed by computer system 62, thus reducing interactions with central server 61 and computer system 60. This is depicted by Figure 6:

⁶⁴⁰ Roy Figure 5A and 5:17-49 (highlighting added).

⁶⁴¹ Roy; Figure 5A.

⁶⁴² Roy; Figure 5A.

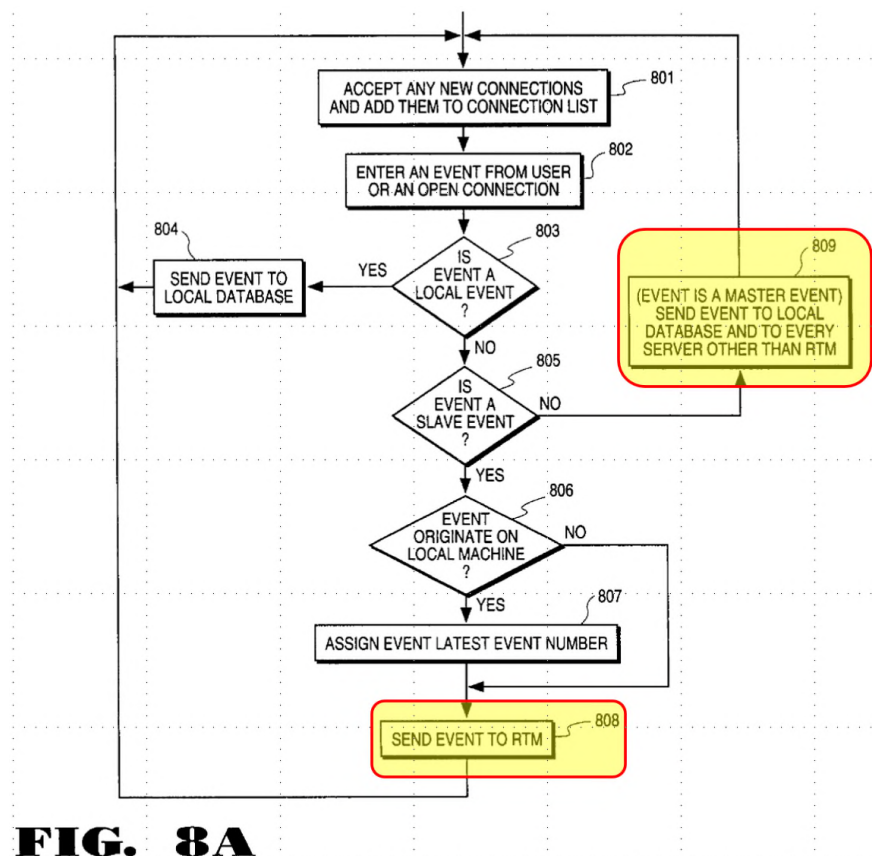
⁶⁴³ Roy; 5:40-43.

⁶⁴⁴ Roy; 6:5-14.

running in a network in which a slave event server has established a direct connection to the master event server.”⁶⁴⁵

502. As another example, Roy further discloses how the client computers and server computer transmit/receive event data to/from one another (i.e., how the client accesses the online multiplayer game application at the server):

“A method is provided of reducing network latency in accessing a database in a network. The network has a number of nodes, including a first node and a second node. The first node includes a first event server for sequencing commands to the database, which is stored in one of the nodes in the network. The second node includes a second event server for sequencing commands to the database. The method comprises the steps of changing the status of the first event server from master event server to slave event server based on a network latency criterion, and assigning the status of master event server to the second event server based on the network latency criterion.”⁶⁴⁶



⁶⁴⁵ Roy; 2:38-41.

⁶⁴⁶ Roy; 1:66-2:10 (highlighting added).

“FIG. 8A illustrates a routine used by a slave event server for opening connections to other event servers and for routing events. Initially, the slave event server accepts any new connections and adds them to its connection list (step 801). Next, the slave event server inputs an event received either from a user or from one of its open connections (step 802). If (in step 803) the event is a local event (i.e., if the event is intended only for use on the local machine), then the slave event server sends the event only to its local database (step 804) and the routine repeats from the beginning. If the event is not a local event and the event is a slave event (step 805) which originated on the local machine (step 806), then the event is assigned the latest event number for the local machine (step 807) and is routed to the RTM (step 808). . . . If the event is neither a local event nor a slave event, then the event is a master event, in which case the event is routed to the local database and to every server connection other than the RTM (step 809).”⁶⁴⁷

[1A] configuring the server to store a model associated with the application and to execute view-generating and controller logic associated with the application; and

503. In my opinion, Roy discloses configuring the server (e.g., central server 61 and/or the computer with the master event server and master database) to store a model associated with the application (e.g., a database representing the world model for the game) and to execute view-generating and controller logic associated with the application, as those terms have been construed. Thus, in my opinion (and as explained below), Roy discloses:

- a. configuring the server (e.g., central server 61 or the computer with the master event server and master database) to store the data, rules, and algorithms affecting the data associated with the application (e.g., a database representing the world model for the game);
- b. configuring the server (e.g., central server 61 or the computer with the master event server and master database) to execute program code (e.g., program code known to a person with ordinary skill in the art at the time the '719 Patent was filed) that processes user requests (e.g., a user enters a command and generates an “event”) and causes the model to be changed and/or the view to be refreshed (e.g., the event causes the database representing the world model to be changed and the user’s screen is ultimately refreshed to reflect the result of their command); and
- c. configuring the server (e.g., central server 61 and/or the computer with the master event server and master database) to execute program code (program code known to a person with ordinary skill in the art at the time the '719 Patent was filed) that generates a screen or window representation of a subset of the model that the application chooses to

⁶⁴⁷ Roy; Figure 8A (annotated, highlighting added) and 6:46-67 (highlighting added).

display (e.g., the server generates an updated view of the game world as affected by the user's command).

504. First, Roy discloses configuring the server to store a model associated with the application. For example, Roy configures the “master database” on the server to store a “database represent[ing] a world model for the game.”⁶⁴⁸ Roy further explains how the “game world can be modeled by a database” which is located on a “central server.”⁶⁴⁹

“A method and apparatus for reducing network latency during execution of a multiple-player game across a computer network are provided. **A master database represents a world model for the game**, and a master event server sequences user inputs, or events, for updating the master database. The master event server and the master database are initially located on a particular node in the network. Every other node that is used by a player in the game has a slave event server and a slave database.”⁶⁵⁰

505. Second, Roy discloses configuring the server to execute view-generating logic associated with the application. As an initial matter, Roy discusses the concept of “[r]endering commands” that “*are [] sent from the updated database 53 in the central server 51 across the network to update the displays 55 and 57 of client computers 50 and 52, respectively.*”⁶⁵¹ Roy discloses that the central server 61 sends rendering commands to one or more client computers (e.g., computers 60 and 62).⁶⁵² While the *transmission* of rendering commands is primarily discussed in relation to the prior art embodiment illustrated in Figures 4A and 4B, Roy acknowledges that “[b]y providing slave copies of the database in every other machine involved in the game, the need to send rendering commands across the network is **reduced** or eliminated, which reduces bandwidth consumption.”⁶⁵³ Thus, while the invention disclosed in Roy may **reduce**

⁶⁴⁸ Roy; Abstract, 4:8-40.

⁶⁴⁹ Roy; 1:28-30.

⁶⁵⁰ Roy; Abstract (highlighting added).

⁶⁵¹ Roy; Figure 4B and 4:23-29 (emphasis added).

⁶⁵² Roy; Figure 4B and 4:23-29.

⁶⁵³ Roy; 4:50-54.

transmitting rendering commands across the network, that necessarily means some rendering commands are still transmitted (and therefore the server still executes view-generating logic).

506. For this discussion, I will focus on rendering commands that are not transmitted (i.e., where the view-generating logic that produces the rendering commands is executed on the same computer where the view is to be displayed). The most complete description of this functionality is provided in relation to Figure 5A, where, focusing on computer 62, Roy explains that “[t]he display 74 of computer system 62 is updated by rendering commands 73 from the slave copy 72 of master database 71.”⁶⁵⁴ These rendering commands (views) are generated by program code in slave copy 72. As Roy explains:

“FIG. 5A illustrates a technique according to the present invention for implementing a multi-user game over the Internet. In FIG. 5A, the game is played by two players, players A and B, operating computer systems 60 and 62, respectively. It is noted, however, that the present invention is equally applicable to a game played simultaneously by more than two players. Initially, the master database 71 and the master event server 66 are located in a central server computer system 61. Located in computer system 60 (used by player A) are a slave event queue 65 and a slave copy 68 of master database 71. Similarly, located in computer system 62 (used by player B) is a slave event server 67 and a slave copy 72 of master database 71. The display 74 of computer system 62 is updated by rendering commands 73 from the slave copy 72 of master database 71, rather than from the master database 71 itself, thereby reducing bandwidth consumption on the network. Similarly, the display 70 associated with computer system 60 is updated by rendering commands 69 from slave copy 68 of master database 71, rather than directly from master database 71. Events 64 from computer system 62 are received by slave event server 67 and then passed along to the master event server 66, unless the events 64 are purely local events for updating only the slave copy 72 of the database. Purely local events might be, for example, events which set display options or other functions that are only relevant to computer system 62. Similarly, events 63 originating from computer system 60 are received first by slave event server 65 and then passed along to master event server 66, unless those events are purely local to computer system 60. Events received by the master event server 66 are used to update the master database 71 and are passed along to all slave event servers as master events 75.”⁶⁵⁵

507. Roy also discloses that whenever a computer becomes an effective server (i.e., the master event server and master database are migrated to that computer, thus resulting in that

⁶⁵⁴ Roy; 5:29-32.

⁶⁵⁵ Roy, 5:17-49.

computer being the “server” of the claim), then that computer executes view-generating logic to generate the views for the player of that computer.⁶⁵⁶ For example, in Figure 6, computer 62 has become an effective **server** and contains the master event server and the master database.⁶⁵⁷ It receives events from player B on computer 62 (and from other players), updates master database 72, and generates a view (as reflected in rendering commands 73) to display on screen 74 of computer 62.⁶⁵⁸ This functionality is described in relation to Figure 5A above, and is illustrated in Figure 6 here:

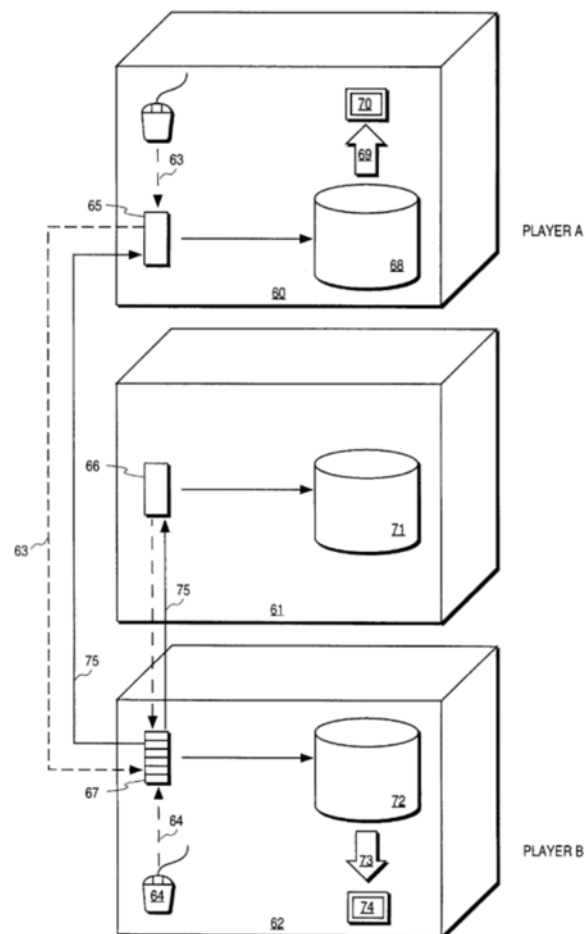


FIG. 6

⁶⁵⁶ Roy; Figure 6 and 4:50-55, 7:1-8.

⁶⁵⁷ Roy; 6:1-11.

⁶⁵⁸ Roy; Figure 6 and 4:50-55, 5:27-36, 7:1-8.

508. To summarize, when the master event server and master database have been migrated to computer 62, computer 62 acts as a server in this client/server system. And computer 62 executes view-generating logic when it creates the rendering commands used to update the display 74.

509. Third, Roy discloses the server executing controller logic associated with the application. For example, Roy explains that the “master event server sequences user inputs, or events, for updating the master database” (i.e., controller logic is executed to update the game world model):

“A method and apparatus for reducing network latency during execution of a multiple-player game across a computer network are provided. A master database represents a world model for the game, and a master event server sequences user inputs, or events, for updating the master database. The master event server and the master database are initially located on a particular node in the network. Every other node that is used by a player in the game has a slave event server and a slave database.”⁶⁵⁹

510. As background, Roy discloses how the user can issue commands and thereby generate “events” using a mouse, joystick, trackball, keyboard, or any other I/O device.⁶⁶⁰ Then, as depicted by Figure 4B, software 54 on the central server 51 “receives events 58 from computers 50 and 52 and provides these events to the database 53 in order to update the game world” and, as described, send rendering commands back to client computers 50 and 52 to reflect the result of the user’s command (i.e., the server executes controller logic to update the game world model stored in database 53 and causes the view to be refreshed on client computers 50 and 52).⁶⁶¹

511. Continuing with the example I discuss above in relation to Figure 6, computer 62 has become an effective server and receives events from player B on computer 62. Computer 62 then executes controller logic to update master database 72 and cause rendering commands 73 to

⁶⁵⁹ Roy; Abstract (highlighting added).

⁶⁶⁰ Roy; 3:66-4:13.

⁶⁶¹ Roy; 4:23-29.

be rendered and shown to the user on his display 74.⁶⁶² Similarly, player B may issue commands that would be processed entirely locally by effective server computer 62. In this case, computer 62 executes controller logic to update master database 72 and cause rendering commands 73 to be rendered and shown to the user on his display 74. In both of these cases, because computer 62 is acting as the server in the client/server system (since the master event server and master database are migrated to that computer), Roy discloses configuring the server to store and execute controller logic.

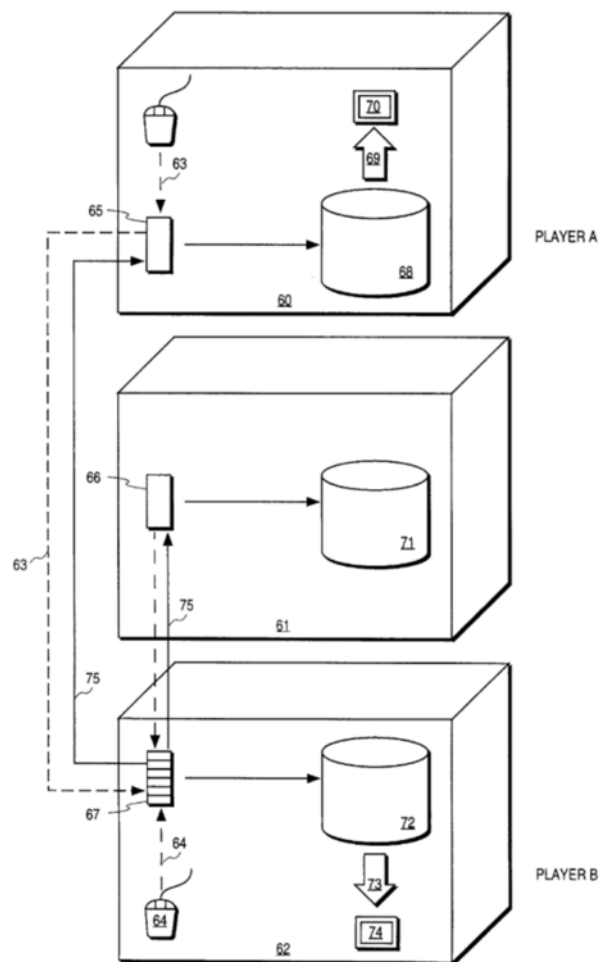


FIG. 6

⁶⁶² Roy; Figure 6 and 5:27-49.

[REDACTED]

Obvious over Roy in light of Bennett⁶⁶³

512. To the extent Roy does not expressly disclose this limitation, it is further my opinion that it would have been obvious to a person of ordinary skill in the art at the time the '719 Patent was filed to modify the online multiplayer gaming system described in Roy based on Bennett's teachings about building distributed applications using objects.

513. A POSITA would have been motivated to modify the interactive online applications described in Roy based on Bennett's teachings. For example, since Smalltalk-80 represented the canonical implementation of the Model-View-Controller (MVC) paradigm, a POSITA would have been motivated to utilize a distributed implementation of Smalltalk such as Bennett when implementing any distributed system using the MVC paradigm.

514. Further, Roy and Bennett are in the same field of endeavor (e.g., distributed applications).

515. In addition, a paper entitled "The Design and Implementation of Distributed Smalltalk"⁶⁶⁴ was published in the Proceedings of the 1987 ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA). A POSITA working in the field of distributed object-based systems who was a member of the Association for Computing Machinery (ACM) – the largest organization of computer scientists and engineers in the world – would have likely received by mail a printed copy of the OOPSLA conference proceedings in which this paper was published. Joining the ACM was a typical step in becoming a professional computer scientist or engineer, and joining SIGPLAN (the Special Interest Group on Programming

⁶⁶³ J.K. Bennett, *Experience with Distributed Smalltalk*, Software—Practice and Experience, Vol. 20(2), pp. 157-180, Feb. 1990 ("Bennett").

⁶⁶⁴ J.K. Bennett, *The Design and Implementation of Distributed Smalltalk*, Proceedings of the Second ACM Conference on Object-Oriented Programming Systems, Languages, and Applications, Orlando, Florida, October 1987, Special Issue of SIGPLAN Notices, 22, (12), pp. 318-330 (1987).

[REDACTED]

Languages) would have been a common step for those aspiring to work in distributed object-oriented systems. All members of SIGPLAN received a printed copy of the OOPSLA conference proceedings as a membership benefit.

516. A POSITA examining Roy and the online multiplayer gaming system described therein would be motivated to modify the client/server architecture of a game depending on the specific requirements for that game.

517. A POSITA examining Roy and the online gaming system described therein would be motivated to modify the client/server architecture of the game depending on the specific requirements for that game. For example, a POSITA would understand that certain types or genres of games would require a different architecture design depending on their processing requirements. As such, a POSITA would seek to distribute the program data and logic effectively and efficiently between the server and client.

518. Bennett discloses a method that enables an application designer to distribute objects containing data and logic (including view-generating logic and controller logic) between servers and client devices, and a POSITA would be motivated to use Bennett's methods.

519. For example, Distributed Smalltalk "provides communication and interaction among geographically remote Smalltalk users, direct access to remote objects, the ability to construct distributed applications in the Smalltalk environment, and a degree of object sharing among users."⁶⁶⁵ "Potential applications of Distributed Smalltalk include mail systems, remote computation servers, remote file servers, and collaborative software development."⁶⁶⁶ "The

⁶⁶⁵ Bennett; p. 157.

⁶⁶⁶ Bennett; p. 157.

distributed aspects of the system are largely user transparent and preserve the reactive quality of Smalltalk objects.”⁶⁶⁷

520. As another example, as described by the '719 Patent, applications can be designed “in terms of the MVC (Model-View-Controller) paradigm” by using the programming language Smalltalk.⁶⁶⁸ “Smalltalk is an object-oriented programming language and highly interactive uniformly object-structured programming environment.”⁶⁶⁹

521. The '719 Patent incorporates by reference “A cookbook for using the model-view controller user interface paradigm in Smalltalk-80”⁶⁷⁰ by Glenn E. Krasner and Stephen T. Pope.⁶⁷¹ This paper was subsequently revised slightly and published as “A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System” (“Pope”). In terms of content, the two papers are nearly identical. “Pope,” as used in this report, refers to the revised (and easier to read) version of the paper. Pope teaches about “the Model-View-Controller (MVC) programming paradigm and methodology used in the Smalltalk-80 programming system.”⁶⁷² Bennett used Smalltalk-80 as its implementation language. “The [object-oriented programming] of these classes are part of the Smalltalk-80 . . . a registered trademark of ParcPlace Systems.”⁶⁷³ Pope describes the benefits of using Smalltalk: “The user interface of the Smalltalk-80 programming environment . . . was developed using a particular strategy of representing information, display, and control.”⁶⁷⁴ “This strategy was chosen to satisfy two goals: (1) to create the special set of system components needed to support a highly interactive software development process, and (2) to provide a general

⁶⁶⁷ Bennett; p. 157.

⁶⁶⁸ '719 Patent, 1:34-35, Background.

⁶⁶⁹ Bennett; p. 157.

⁶⁷⁰ G. Krasner and S. Pope, *A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80*; Journal of Object-Oriented Programming, SIGS Publications, Vol. 1, No. 3, pp. 26-49, Aug. 1988.

⁶⁷¹ '719 Patent, 1:35-39.

⁶⁷² Pope; Abstract, p. 1.

⁶⁷³ Bennett; p. 161.

⁶⁷⁴ Pope; p. 2.

set of system components that make it possible for programmers to create portable interactive graphical applications easily.”⁶⁷⁵ Pope further describes the benefits of MVC programming. For example, see the excerpt from Pope shown below:⁶⁷⁶

MVC and the Issues of Reusability and Pluggability

When building interactive applications, as with other programs, modularity of components has enormous benefits. Isolating functional units from each other as much as possible makes it easier for the application designer to understand and modify each particular unit, without having to know everything about the other units. Our experiences with the Smalltalk-76 programming system showed that one particular form of modularity--a three-way separation of application components--has payoff beyond merely making the designer's life easier. This three-way division of an application entails separating (1) the parts that represent the model of the underlying application domain from (2) the way the model is presented to the user and from (3) the way the user interacts with it.

Model-View-Controller (MVC) programming is the application of this three-way factoring, whereby objects of different classes take over the operations related to the application domain (the model), the display of the application's state (the view), and the user interaction with the model and the view (the controller). In earlier Smalltalk system user interfaces, the tools that were put into the interface tended to consist of arrangements of four basic viewing idioms: paragraphs of text, lists of text (menus), choice "buttons," and graphical forms (bit- or pixel-maps). These tools also tended to use three basic user interaction paradigms: browsing, inspecting and editing. A goal of the current Smalltalk-80 system was to be able to define user interface components for handling these idioms and paradigms once, and share them among all the programming environment tools and user-written applications using the methodology of MVC programming.

522. Bennett's Distributed Smalltalk "is an implementation of Smalltalk."⁶⁷⁷ Thus, because a POSITA would understand that Smalltalk could be used to design applications using an MVC paradigm, a POSITA would similarly understand that Distributed Smalltalk could be used to design distributed applications using an MVC paradigm.

⁶⁷⁵ Pope; p. 2

⁶⁷⁶ Pope; p. 2.

⁶⁷⁷ Bennett; p. 157.

[REDACTED]

523. At least in light of the foregoing, a POSITA would be motivated to create and/or modify the online interactive applications described by Roy by using Distributed Smalltalk. For example, a POSITA may seek to “offload[] a compute-intensive task” to the server.⁶⁷⁸ Similarly, a POSITA could distribute DS objects containing data and logic (including model data, view-generation logic, or controller logic) between the client and server depending on the requirements for a specific application, and would understand that the application could be implemented per the MVC paradigm in view of Bennett’s use of Smalltalk. This is particularly true since Roy was equally concerned with distributing processing between the computer systems.⁶⁷⁹

[1B] configuring the client to store at least a subset of the model associated with the application and to execute at least a subset of the view-generating and controller logic associated with the application;

524. In my opinion, Roy discloses configuring the client (e.g., computer system 60) to store at least a subset of the model associated with the application (e.g., a subset of the database comprising the world model for the online multiplayer application) and to execute at least a subset of the view-generating and controller logic associated with the application, as those terms have been construed. Thus, in my opinion (and as explained below), Roy discloses:

- a. configuring the client (e.g., computer system 60) to store at least a subset of the data, rules, and algorithms affecting the data associated with the application (e.g., at least a subset of the world model stored in the master database is migrated to each client computer);
- b. configuring the client (e.g., computer system 60) to execute program code (program code known to a POSITA at the time the ’719 Patent was filed) that processes user requests (e.g., a user command that generates an “event”) and causes the model to be changed and/or the view to be refreshed (e.g., the event causes the local database containing a subset of the game world data to be changed and the view is refreshed on computer system 60 to reflect the change);
- c. configuring the client (e.g., computer system 60) to execute program code (program code known to a POSITA at the time the ’719 Patent was filed) that generates a screen

⁶⁷⁸ Bennett; p. 157.

⁶⁷⁹ Roy; 4:18-54.

or window representation of a subset of the model that the application chooses to display (e.g., client computer system 60 generates an updated view that is displayed to the user of computer system 60); and

- d. that the view-generating logic and controller logic executed on the client is the same view-generating logic and controller logic executed on the server, or some subset thereof (e.g., the *same* view-generating logic and controller logic would be executed on each computer system connected to the game network).

525. First, Roy discloses configuring the client to store at least a subset of the model associated with the application. In particular, Roy discloses that “[s]lave copies of the master database (or portions thereof) are stored” on each client computer.⁶⁸⁰ As I explained previously, the databases of Roy represent the world model for the game. So because the clients contain a copy of the master database (or a portion thereof), it necessarily means that the client is configured to store at least a subset of the model. Further, as Roy explains:

“A method and apparatus for reducing network latency during execution of a multiple-player game across a computer network are provided. A master database represents a world model for the game, and a master event server sequences user inputs, or events, for updating the master database. The master event server and the master database are initially located on a particular node in the network. Every other node that is used by a player in the game has a slave event server and a slave database.”⁶⁸¹

⁶⁸⁰ Roy; 4:44-48.

⁶⁸¹ Roy; Abstract (highlighting added).

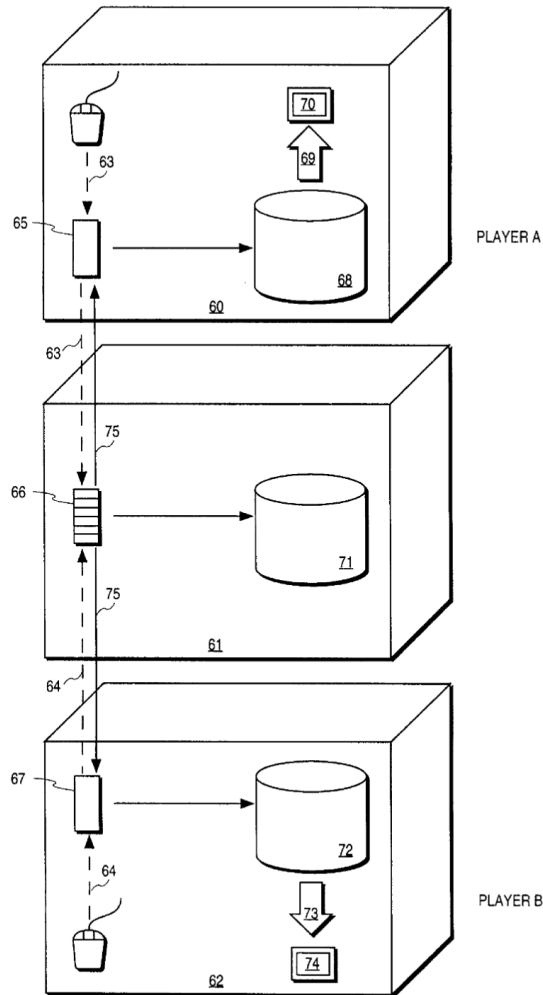


FIG. 5A

“FIG. 5A illustrates a technique according to the present invention for implementing a multi-user game over the Internet. In FIG. 5A, the game is played by two players, players A and B, operating computer systems 60 and 62, respectively. It is noted, however, that the present invention is equally applicable to a game played simultaneously by more than two players. Initially, the master database 71 and the master event server 66 are located in a central server computer system 61. Located in computer system 60 (used by player A) are a slave event queue 65 and a slave copy 68 of master database 71. Similarly, located in computer system 62 (used by player B) is a slave event server 67 and a slave copy 72 of master database 71. The display 74 of computer system 62 is updated by rendering commands 73 from the slave copy 72 of master database 71, rather than from the master database 71 itself, thereby reducing bandwidth consumption on the network. Similarly, the display 70 associated with computer system 60 is updated by rendering commands 69 from slave copy 68 of master database 71, rather than directly from master database 71. Events 64 from computer system 62 are received by slave event server 67 and then passed along to the master event server 66, unless the events 64 are purely local events for updating only the slave copy 72 of the database. Purely local events might be, for example, events which set display options or other functions that are only relevant to computer system 62. Similarly, events 63 originating from computer system 60 are



received first by slave event server 65 and then passed along to master event server 66, unless those events are purely local to computer system 60. Events received by the master event server 66 are used to update the master database 71 and are passed along to all slave event servers as master events 75.”⁶⁸²

526. Thus, as an example, the entire master database “or portions thereof”⁶⁸³ would be saved in database 68 of client computer 60, meaning at least a subset of the data comprising the world model would be stored on the client. (This is true even when the master database and master event server have been migrated to computer 62, which is the scenario I discuss in claim element 1A.)

527. Second, Roy discloses configuring the client to execute at least a subset of the view-generating logic associated with the application. As an example, Roy discloses that the display 70 of client computer 60 is updated by rendering commands 69 from the slave copy 68 of master database 72 (when the master database has been migrated to computer 62), rather than from the master database itself, thereby reducing bandwidth consumption on the network:

⁶⁸² Roy; Figure 5A and 5:17-49 (highlighting added).

⁶⁸³ Roy; 4:47-48.

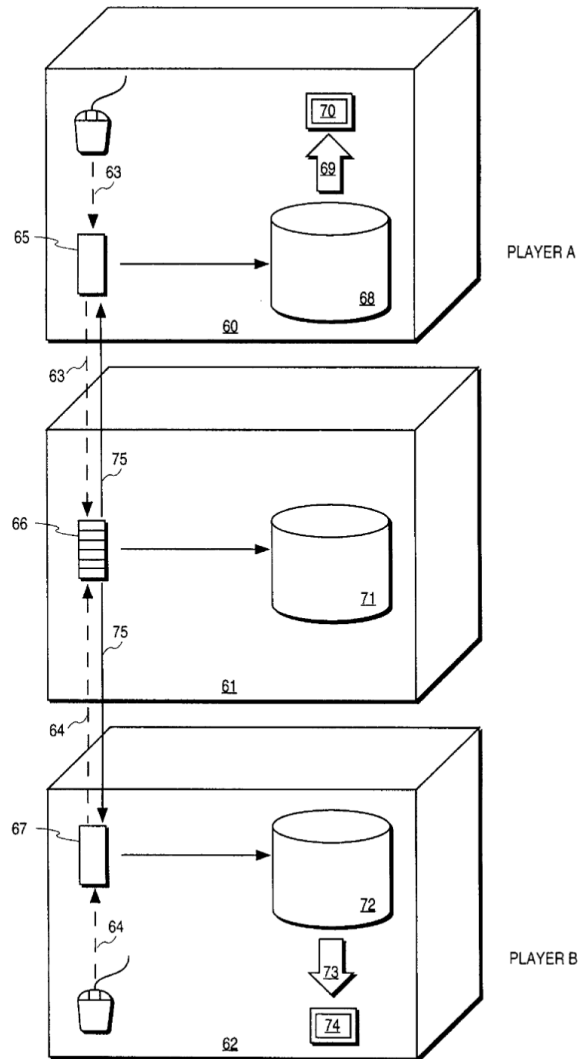


FIG. 5A

“FIG. 5A illustrates a technique according to the present invention for implementing a multi-user game over the Internet. In FIG. 5A, the game is played by two players, players A and B, operating computer systems 60 and 62, respectively. It is noted, however, that the present invention is equally applicable to a game played simultaneously by more than two players. Initially, the master database 71 and the master event server 66 are located in a central server computer System 61. Located in computer system 60 (used by player A) are a slave event queue 65 and a slave copy 68 of master database 71. Similarly, located in computer system 62 (used by player B) is a slave event server 67 and a slave copy 72 of master database 71. The display 74 of computer system 62 is updated by rendering commands 73 from the slave copy 72 of master database 71, rather than from the master database 71 itself, thereby reducing bandwidth consumption on the network. Similarly, the display 70 associated with computer system 60 is updated by rendering commands 69 from slave copy 68 of master database 71, rather than directly from master database 71. Events 64 from computer system 62 are received by slave event server 67 and then passed along to the master event server 66, unless the events 64 are purely local events for updating only the slave copy 72 of the database. Purely local

events might be, for example, events which set display options or other functions that are only relevant to computer system 62. Similarly, events 63 originating from computer system 60 are received first by slave event server 65 and then passed along to master event server 66, unless those events are purely local to computer system 60. Events received by the master event server 66 are used to update the master database 71 and are passed along to all slave event servers as master events 75.”⁶⁸⁴

528. As a specific example, Roy discloses client computer system 60 executing at least a subset of the view-generating logic. As described above, slave copies of the database (or a portion thereof) are migrated to every client computer because this reduces “the need to send rendering commands across the network.”⁶⁸⁵ Thus, each client computer may execute view-generating logic to create a view for each respective player because each player has at least a subset of the world model stored on their respective computers.⁶⁸⁶ Further, it would be understood to a POSITA, or at least rendered obvious, that the view-generating logic would be the same on each computer for efficiency and economic reasons. It would be particularly obvious to a POSITA given that Roy specifically addresses online multiplayer games wherein each player sees the *same* world. It would not make sense to use different logic to generate views of the same world for each computer.

529. As just one simple example, imagine two players connected to an online multiplayer game as described by Roy. Each controls an avatar that can explore the virtual world. If, for example, the two players were standing in the exact same location and were looking at the virtual world from the same perspective, then each player’s screen would display the same view. In this case, it would make little sense if the view-generating logic on each player’s computer was different. A POSITA would not endeavor to do redundant work and coding for the same functionality.

⁶⁸⁴ Roy; Figure 5A and 5:17-49 (highlighting added).

⁶⁸⁵ Roy; 4:51-54.

⁶⁸⁶ Roy; 5:29-33 (“The display 74 of computer system 62 is updated by rendering commands 73 from the slave copy 72 of master database 71, rather than from the master database 71 itself.”).



530. That the program code on each computer would be the same (or at least that it would be obvious) is also evident by the very nature of commercial multi-player games. The program code for the game running on player A's computer is the same as the program code for the game running player B's computer. This is particularly true where, as Roy explains, any node may become an effective server—each player's computer would need the functionality to run the game (and serve as the “master”). As such a person of ordinary skill in the art would understand (or at the very least find it obvious) that program code used to operate the game (e.g., generate the rendering commands for display) would be the same across the player's computers.

531. Third, Roy discloses configuring the client to execute at least a subset of the controller logic associated with the application on the client. For example, Roy discloses that the client executes controller logic to process local events and cause the model of the world stored in the local database to be updated and the view to be refreshed for that user. Roy explains:

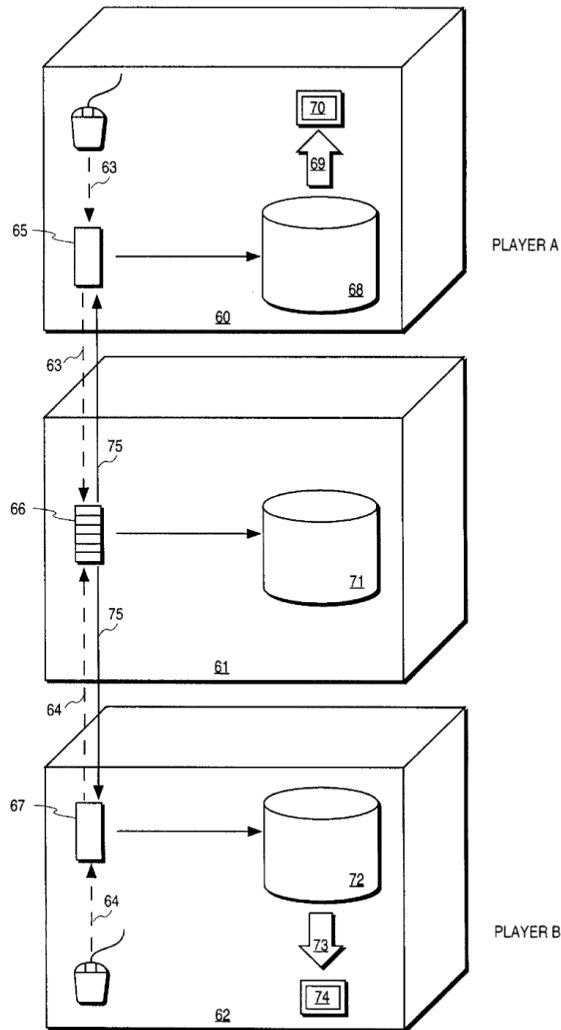


FIG. 5A

“FIG. 5A illustrates a technique according to the present invention for implementing a multi-user game over the Internet. In FIG. 5A, the game is played by two players, players A and B, operating computer Systems 60 and 62, respectively. It is noted, however, that the present invention is equally applicable to a game played simultaneously by more than two players. Initially, the master database 71 and the master event server 66 are located in a central server computer system 61. Located in computer system 60 (used by player A) are a slave event queue 65 and a slave copy 68 of master database 71. Similarly, located in computer system 62 (used by player B) is a slave event server 67 and a slave copy 72 of master database 71. The display 74 of computer system 62 is updated by rendering commands 73 from the slave copy 72 of master database 71, rather than from the master database 71 itself, thereby reducing bandwidth consumption on the network. Similarly, the display 70 associated with computer system 60 is updated by rendering commands 69 from slave copy 68 of master database 71, rather than directly from master database 71. Events 64 from computer system 62 are received by slave event server 67 and then passed along to the master event server 66, unless the events 64 are purely local events for updating only the slave copy 72 of the database. Purely local events might be, for example, events which set display options or other functions that are only

relevant to computer system 62. Similarly, events 63 originating from computer system 60 are received first by slave event server 65 and then passed along to master event server 66, unless those events are purely local to computer system 60. Events received by the master event server 66 are used to update the master database 71 and are passed along to all slave event servers as master events 75.”⁶⁸⁷

532. As a specific example, Roy discloses that client computer system 60 executes at least a subset of the controller logic. As described, client computer 60 processes events generated by user commands and determines whether the events should be passed along to the server, or whether they are “local events” that are used to update the local model stored in database 68 and cause display 70 to be refreshed.⁶⁸⁸ This determination is done by an event server 65 on computer system 60. Every computer system connected to the game has an “event server” and each is tasked with “routing events to the local database and to other event servers.”⁶⁸⁹ And the master event server can be moved between the central server 61 and any computer system.⁶⁹⁰ Thus, the *same* controller logic can be executed on any client computer.

533. Additionally, for the same reasons described above regarding the view-generating logic, it would be understood by a POSITA, or would at least be obvious to a POSITA, that the controller logic on each computer system would be the same. For example, it would be obvious to a POSITA that each computer would process “purely local events” in the same way, such as local events for setting display options.⁶⁹¹ Further, because any computer can become an effective server, a POSITA would understand that the controller logic on each computer would be the same, or at the very least it would be obvious to do so for the reasons discussed above.

⁶⁸⁷ Roy; Figure 5A and 5:17-49 (highlighting added).

⁶⁸⁸ Roy; 5:17-49.

⁶⁸⁹ Roy; 4:62-66.

⁶⁹⁰ Roy; 6:1-11.

⁶⁹¹ Roy; 5:40-43.

[REDACTED]

Obvious over Roy in light of Rover⁶⁹²

534. To the extent Roy does not expressly disclose this limitation, it is further my opinion that it would have been obvious to a person of ordinary skill in the art at the time the '719 Patent was filed to modify the online multiplayer gaming system described in Roy based on Rover's teachings about moving application data and logic between a client and server using relocatable dynamic objects ("RDOs").

535. A POSITA would have been motivated to modify the online gaming system described in Roy based on Rover's teachings. For example, Roy and Rover are in the same field of endeavor (interactive distributed systems).

536. In addition, Rover was published in the leading venue for operating and distributed systems research (the annual Symposium on Operating Systems Principles (SOSP). A POSITA working in the field of distributed systems who was a member of the Association for Computing Machinery (ACM) – the largest organization of computer scientists and engineers in the world – would have likely received by post mail a printed copy of the SOSP conference proceedings in which Rover was published.

537. Joining the ACM was a typical step in becoming a professional computer scientist or engineer, and joining SIGOPS (the Special Interest Group on Operating Systems) would have been a common step for those aspiring to work in distributed systems. All members of SIGOPS received a printed copy of the SOSP conference proceedings as a membership benefit.

538. A POSITA examining Roy and the online gaming system described therein would be motivated to modify the client/server architecture of the game depending on the specific requirements for that game. For example, a POSITA would understand that certain types or genres

⁶⁹² A. Joseph et al., *Rover: A Toolkit for Mobile Information Access*, Proc. 15th ACM SOSP, pp. 156-171, Dec. 1995.

[REDACTED]

of games would require a different architecture design depending on their processing requirements. As such, a POSITA would seek to distribute the program data and logic effectively and efficiently between the server and client. Rover discloses a client/server architecture that utilizes RDOs to move logic between the server and client as desired by the application designer.

539. For example, Rover explains that “[t]he Rover toolkit combines relocatable dynamic objects and queued remote procedure calls to provide unique services for ‘roving’ mobile applications.”⁶⁹³ “A relocatable dynamic object is an object with a well-defined interface that can be dynamically loaded into a client computer from a server computer (or vice versa) to reduce client-server communication requirements.”⁶⁹⁴ “The Rover toolkit also offers advantages for workstation applications by providing a uniform distributed object architecture for code shipping, object caching, and asynchronous object invocation.”⁶⁹⁵ “Depending on the power of the mobile host and the available bandwidth, Rover dynamically adapts and moves functionality between the client and the server.”⁶⁹⁶ “By migrating a GUI to the client, input events can be processed locally, even if the client becomes disconnected.”⁶⁹⁷

540. In light of at least the foregoing, a POSITA would understand that data and application logic (including view-generating and controller logic) for the online gaming system described in Roy could be allocated to either the server or the client, and would be motivated to incorporate Rover’s method for distributing that data and logic (including model data, controller logic, and view-generating logic) because, for example, “[d]epending on the power of the [client]

⁶⁹³ Rover; p. 156.

⁶⁹⁴ Rover; p. 156.

⁶⁹⁵ Rover; p. 156.

⁶⁹⁶ Rover; p. 158.

⁶⁹⁷ Rover; p. 159.

[REDACTED]

and the available bandwidth, Rover dynamically adapts and moves functionality between the client and the server.”⁶⁹⁸

Obvious over Roy in light of Bennett⁶⁹⁹

541. To the extent Roy does not expressly disclose this limitation, it is further my opinion that it would have been obvious to a person of ordinary skill in the art at the time the ’719 Patent was filed to modify the online multiplayer gaming system described in Roy based on Bennett’s teachings about building distributed applications using objects.

542. A POSITA would have been motivated to modify the interactive online applications described in Roy based on Bennett’s teachings. For example, since Smalltalk-80 represented the canonical implementation of the Model-View-Controller (MVC) paradigm, a POSITA would have been motivated to utilize a distributed implementation of Smalltalk such as Bennett when implementing any distributed system using the MVC paradigm.

543. Further, Roy and Bennett are in the same field of endeavor (e.g., distributed applications).

544. In addition, a paper entitled “The Design and Implementation of Distributed Smalltalk”⁷⁰⁰ was published in the Proceedings of the 1987 ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA). A POSITA working in the field of distributed object-based systems who was a member of the Association for Computing Machinery (ACM) – the largest organization of computer scientists and engineers in the world – would have likely received by mail a printed copy of the OOPSLA conference proceedings in

⁶⁹⁸ Rover; p. 158.

⁶⁹⁹ J.K. Bennett, *Experience with Distributed Smalltalk*, Software—Practice and Experience, Vol. 20(2), pp. 157-180, Feb. 1990 (“Bennett”).

⁷⁰⁰ J.K. Bennett, *The Design and Implementation of Distributed Smalltalk*, Proceedings of the Second ACM Conference on Object-Oriented Programming Systems, Languages, and Applications, Orlando, Florida, October 1987, Special Issue of SIGPLAN Notices, 22, (12), pp. 318-330 (1987).



which this paper was published. Joining the ACM was a typical step in becoming a professional computer scientist or engineer, and joining SIGPLAN (the Special Interest Group on Programming Languages) would have been a common step for those aspiring to work in distributed object-oriented systems. All members of SIGPLAN received a printed copy of the OOPSLA conference proceedings as a membership benefit.

545. A POSITA examining Roy and the online multiplayer gaming system described therein would be motivated to modify the client/server architecture of a game depending on the specific requirements for that game.

546. A POSITA examining Roy and the online gaming system described therein would be motivated to modify the client/server architecture of the game depending on the specific requirements for that game. For example, a POSITA would understand that certain types or genres of games would require a different architecture design depending on their processing requirements. As such, a POSITA would seek to distribute the program data and logic effectively and efficiently between the server and client.

547. Bennett discloses a method that enables an application designer to distribute objects containing data and logic (including view-generating logic and controller logic) between servers and client devices, and a POSITA would be motivated to use Bennett's methods.

548. For example, Distributed Smalltalk "provides communication and interaction among geographically remote Smalltalk users, direct access to remote objects, the ability to construct distributed applications in the Smalltalk environment, and a degree of object sharing among users."⁷⁰¹ "Potential applications of Distributed Smalltalk include mail systems, remote computation servers, remote file servers, and collaborative software development."⁷⁰² "The

⁷⁰¹ Bennett; p. 157.

⁷⁰² Bennett; p. 157.

distributed aspects of the system are largely user transparent and preserve the reactive quality of Smalltalk objects.”⁷⁰³

549. As another example, as described by the '719 Patent, applications can be designed “in terms of the MVC (Model-View-Controller) paradigm” by using the programming language Smalltalk.⁷⁰⁴ “Smalltalk is an object-oriented programming language and highly interactive uniformly object-structured programming environment.”⁷⁰⁵

550. The '719 Patent incorporates by reference “A cookbook for using the model-view controller user interface paradigm in Smalltalk-80”⁷⁰⁶ by Glenn E. Krasner and Stephen T. Pope.⁷⁰⁷ This paper was subsequently revised slightly and published as “A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System” (“Pope”). In terms of content, the two papers are nearly identical. “Pope,” as used in this report, refers to the revised (and easier to read) version of the paper. Pope teaches about “the Model-View-Controller (MVC) programming paradigm and methodology used in the Smalltalk-80 programming system.”⁷⁰⁸ Bennett used Smalltalk-80 as its implementation language. “The [object-oriented programming] of these classes are part of the Smalltalk-80 . . . a registered trademark of ParcPlace Systems.”⁷⁰⁹ Pope describes the benefits of using Smalltalk: “The user interface of the Smalltalk-80 programming environment . . . was developed using a particular strategy of representing information, display, and control.”⁷¹⁰ “This strategy was chosen to satisfy two goals: (1) to create the special set of system components needed to support a highly interactive software development process, and (2) to provide a general

⁷⁰³ Bennett; p. 157.

⁷⁰⁴ '719 Patent, 1:34-35, Background.

⁷⁰⁵ Bennett; p. 157.

⁷⁰⁶ G. Krasner and S. Pope, *A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80*; Journal of Object-Oriented Programming, SIGS Publications, Vol. 1, No. 3, pp. 26-49, Aug. 1988.

⁷⁰⁷ '719 Patent, 1:35-39.

⁷⁰⁸ Pope; Abstract, p. 1.

⁷⁰⁹ Bennett; p. 161.

⁷¹⁰ Pope; p. 2.

set of system components that make it possible for programmers to create portable interactive graphical applications easily.”⁷¹¹ Pope further describes the benefits of MVC programming. For example, see the excerpt from Pope shown below:⁷¹²

MVC and the Issues of Reusability and Pluggability

When building interactive applications, as with other programs, modularity of components has enormous benefits. Isolating functional units from each other as much as possible makes it easier for the application designer to understand and modify each particular unit, without having to know everything about the other units. Our experiences with the Smalltalk-76 programming system showed that one particular form of modularity--a three-way separation of application components--has payoff beyond merely making the designer's life easier. This three-way division of an application entails separating (1) the parts that represent the model of the underlying application domain from (2) the way the model is presented to the user and from (3) the way the user interacts with it.

Model-View-Controller (MVC) programming is the application of this three-way factoring, whereby objects of different classes take over the operations related to the application domain (the model), the display of the application's state (the view), and the user interaction with the model and the view (the controller). In earlier Smalltalk system user interfaces, the tools that were put into the interface tended to consist of arrangements of four basic viewing idioms: paragraphs of text, lists of text (menus), choice "buttons," and graphical forms (bit- or pixel-maps). These tools also tended to use three basic user interaction paradigms: browsing, inspecting and editing. A goal of the current Smalltalk-80 system was to be able to define user interface components for handling these idioms and paradigms once, and share them among all the programming environment tools and user-written applications using the methodology of MVC programming.

551. Bennett's Distributed Smalltalk "is an implementation of Smalltalk."⁷¹³ Thus, because a POSITA would understand that Smalltalk could be used to design applications using an MVC paradigm, a POSITA would similarly understand that Distributed Smalltalk could be used to design distributed applications using an MVC paradigm.

⁷¹¹ Pope; p. 2

⁷¹² Pope; p. 2.

⁷¹³ Bennett; p. 157.



552. At least in light of the foregoing, a POSITA would be motivated to create and/or modify the online interactive applications described by Roy by using Distributed Smalltalk. For example, a POSITA may seek to “offload[] a compute-intensive task” to the server.⁷¹⁴ Similarly, a POSITA could distribute DS objects containing data and logic (including model data, view-generation logic, or controller logic) between the client and server depending on the requirements for a specific application, and would understand that the application could be implemented per the MVC paradigm in view of Bennett’s use of Smalltalk.

[1C] wherein one or more portions of the application are performed at the client without the client having to interact with the server, and

553. In my opinion, as described above regarding claim preamble 1P, Roy discloses wherein one or more portions of the application (e.g., the online multiplayer game) are performed at the client (e.g., computer system 60) without the client having to interact with the server (e.g., the computer system that has the master event server and master database). I incorporate those explanations here.

554. Roy further discloses:

⁷¹⁴ Bennett; p. 157.

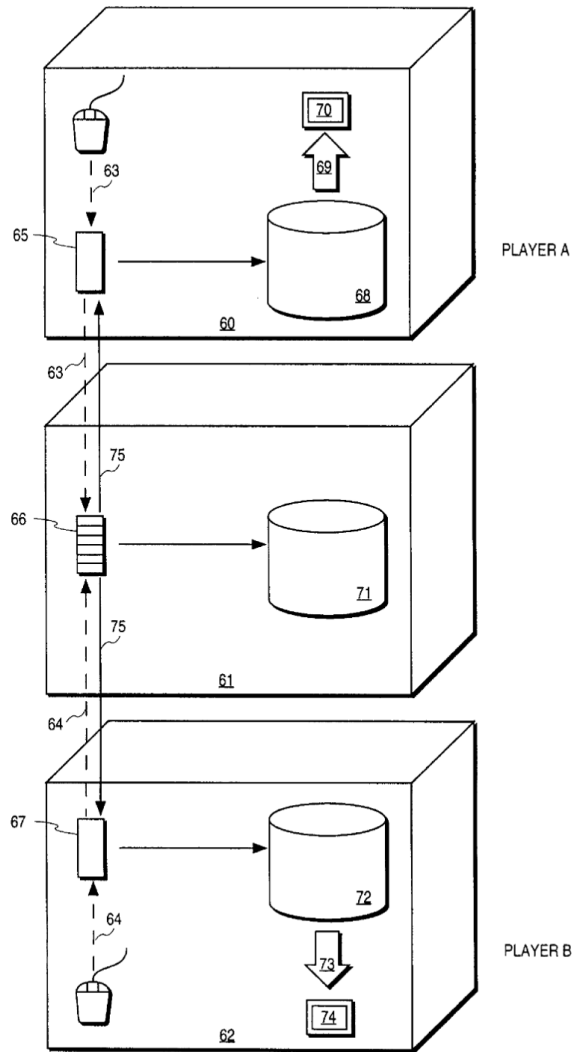
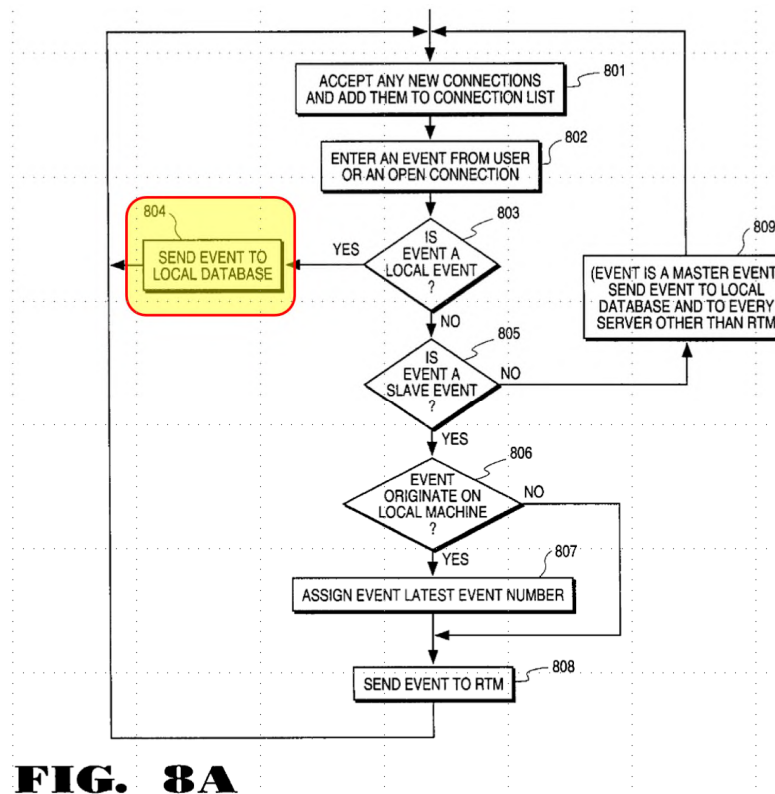


FIG. 5A

“FIG. 5A illustrates a technique according to the present invention for implementing a multi-user game over the Internet. In FIG. 5A, the game is played by two players, players A and B, operating computer systems 60 and 62, respectively. It is noted, however, that the present invention is equally applicable to a game played simultaneously by more than two players. Initially, the master database 71 and the master event server 66 are located in a central server computer system 61. Located in computer system 60 (used by player A) are a slave event queue 65 and a slave copy 68 of master database 71. Similarly, located in computer system 62 (used by player B) is a slave event server 67 and a slave copy 72 of master database 71. The display 74 of computer system 62 is updated by rendering commands 73 from the slave copy 72 of master database 71, rather than from the master database 71 itself, thereby reducing bandwidth consumption on the network. Similarly, the display 70 associated with computer system 60 is updated by rendering commands 69 from slave copy 68 of master database 71, rather than directly from master database 71. Events 64 from computer system 62 are received by slave event server 67 and then passed along to the master event server 66, unless the events 64 are purely local events for updating only the slave copy 72 of the database. Purely local

events might be, for example, events which set display options or other functions that are only relevant to computer system 62. Similarly, events 63 originating from computer system 60 are received first by slave event server 65 and then passed along to master event server 66, unless those events are purely local to computer system 60. Events received by the master event server 66 are used to update the master database 71 and are passed along to all slave event servers as master events 75.”⁷¹⁵



“FIG. 8A illustrates a routine used by a slave event server for opening connections to other event servers and for routing events. Initially, the slave event server accepts any new connections and adds them to its connection list (step 801). Next, the slave event server inputs an event received either from a user or from one of its open connections (step 802). If (in step 803) the event is a local event (i.e., if the event is intended only for use on the local machine), then the slave event server sends the event only to its local database (step 804) and the routine repeats from the beginning. If the event is not a local event and the event is a slave event (step 805) which originated on the local machine (step 806), then the event is assigned the latest event number for the local machine (step 807) and is routed to the RTM (step 808). . . . If the event is neither a local event nor a slave event, then the event is a master event, in which case the event is routed to the local database and to every server connection other than the RTM (step 809)”⁷¹⁶

⁷¹⁵ Roy; Figure 5A and 5:17-49 (highlighting added).

⁷¹⁶ Roy; Figure 8A (annotated, highlighting added) and 6:46-67 (highlighting added).

[REDACTED]

555. As an example, and as described above, “local events” are processed at the client without having to interact with the server.⁷¹⁷

[1D] further wherein the client and the server both locally maintain at least a portion of the model and execute the view-generating and controller logic associated therewith.

556. In my opinion, Roy discloses that the client and the server both locally maintain at least a portion of the model (e.g., a portion of the game world model), and execute the view-generating and controller logic associated therewith, as described above regarding claim elements 1A and 1B. I incorporate those explanations here.

557. I explained in my descriptions of the model, view-generating logic, and controller logic for claim elements 1A and 1B how: (1) both the client and server maintain at least a portion of the model (e.g., a portion of the game world model), (2) the client and server execute the same controller logic (e.g., the client and server execute the same logic to process the same event created by a user command to update their respective local world models and/or to cause their respective screens to be refreshed) and (3) the client and server execute the same view-generating logic (e.g., the client and server computers execute the same view-generating logic when showing a view of the same world to the respective players on each computer). I incorporate those explanations here.

c. Roy Renders Obvious Claim 2 of the '719 Patent

[2] The method of claim 1, wherein the client and server communicate over a HyperText Transport Protocol [HTTP] network.

558. As discussed above, Roy anticipates or renders obvious claim 1 of the '719 Patent.

559. In my opinion, Roy renders obvious wherein the client and server communicate over an HTTP network. For example, as detailed above, Roy discloses a method for solving latency

⁷¹⁷ Roy; Figure 8A and 5:29-46.

issues for multiplayer games that are played *on the Internet*.⁷¹⁸ It would be obvious to a POSITA at the time the '719 Patent was filed that HTTP would (or could) be used for the client/server system described by Roy to communicate over the Internet.

560. Indeed, it would be obvious to a POSITA that *any* client/server system from the relevant time could utilize HTTP, as discussed in Section 8.B.7.a.

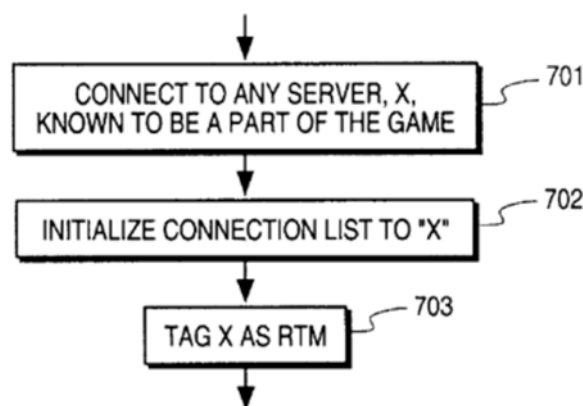
d. Roy Anticipates or Renders Obvious Claim 12 of the '719 Patent

[12] The method of claim 1, wherein the at least a subset of the model, the view-generating and the controller logic associated with the application are downloaded from the server to the client upon demand.

561. As discussed above, Roy anticipates or renders obvious claim 1 of the '719 Patent.

562. In my opinion, Roy discloses or renders obvious wherein the at least a subset of the model, the view-generating and the controller logic associated with the application are downloaded from the server to the client upon demand.

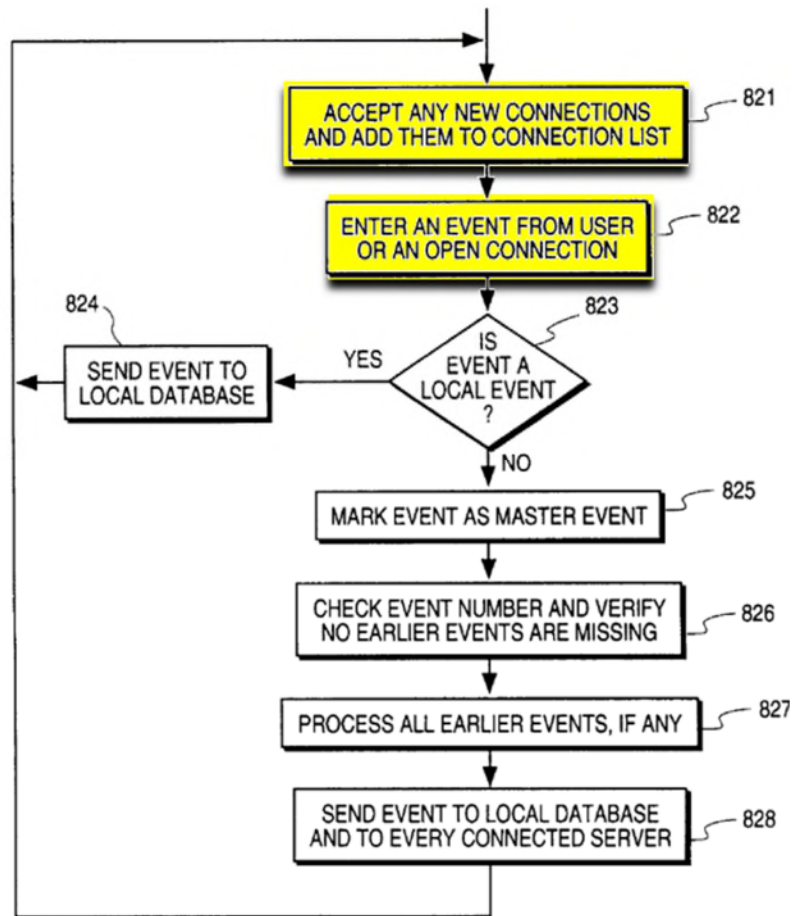
563. To start, Roy discloses how client computers may connect to a server to join the online game. For example, Roy discloses “a technique for establishing a connection to a computer system having multi-player game software.”⁷¹⁹ This is depicted by Figure 7:



⁷¹⁸ Roy; 1:19-22 (“[T]here are unique problems associated with implementing a multi-player computer game across a [WAN], particularly one as expansive as the Internet.”).

⁷¹⁹ Roy; Figure 7 and 2:41-43.

564. Similarly, Figure 8B “illustrates a routine used by a master event server for opening connections.”⁷²⁰



565. As another example, Figure 6 depicts how client computers 60 and 62 access an online application at central server 61. Or, as Roy describes, “FIG. 6 illustrates a multi-player game running in a network in which a slave event server has established a direct connection to the master event server.”⁷²¹

566. Additionally, Roy discloses that server data (including the model) can be migrated to any client computer on demand:

⁷²⁰ Roy; Figure 8B (highlighting added) and 7:1-2.

⁷²¹ Roy; 2:38-41.

“Regardless of which device has the actual authority to approve of migration of the master event server, the migration process can be initiated by a request or command transmitted over the network by a slave event server. The request or command is a special event known as a ‘slave migration event.’ The master event server initiates the migration process by sending a ‘master migration event’. When a slave event server receives a slave migration event from another slave event server, it adds its identifier to the end of the event message. Hence, the slave migration event gets annotated with the complete path it follows to reach the master event server. The master migration event sent in reply also contains the same path information. Accordingly, interceding slave event servers can correctly route the master migration event to the slave event server that transmitted the slave migration event and update their respective RTM information.”⁷²²

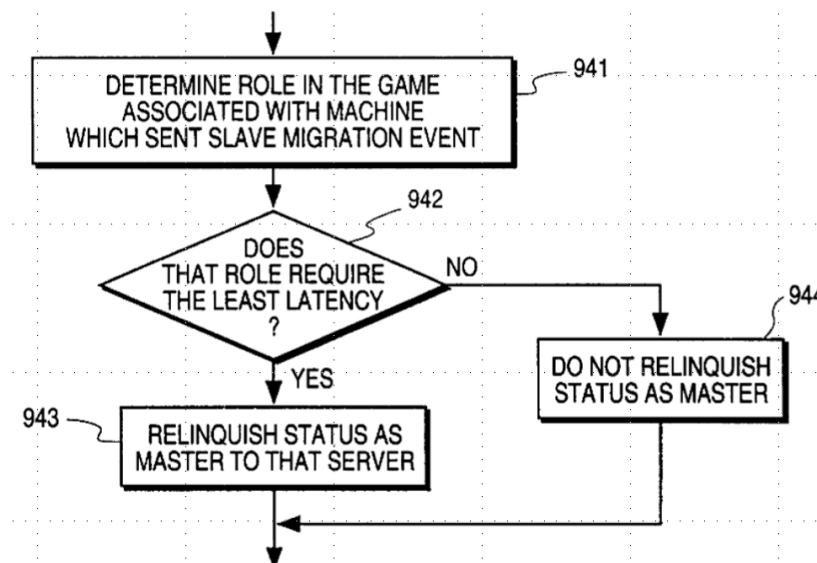


FIG. 9C

“FIG. 9C illustrates a routine by which a master event server can determine whether to relinquish its status as master event server. Upon receiving a slave migration event, the master event server identifies the role in the game that is associated with the slave event server that sent the slave migration event (step 941). If that role requires the lowest latency of any role in the game, then it is appropriate to migrate the master event server. Accordingly, the master event server relinquishes its status as master event server (step 943). Otherwise, the master event server retains its status as master (step 944).”⁷²³

567. In light of the foregoing, it would be understood by or at least rendered obvious to a POSITA that, when a new user joined the online multiplayer game, a slave copy of the database would be transferred to the new user’s computer (i.e., at least a subset of the model would be

⁷²² Roy; 7:43-59 (highlighting added).

⁷²³ Roy; Figure 9C and 8:41-51 (highlighting added).

[REDACTED]

downloaded from the server to the client upon demand). Indeed, *every* client computer obtained a slave copy of the model at some point.⁷²⁴ Roy also explains that the event server of a new player can “connect[] to another event server that is already part of the game.”⁷²⁵

568. It would similarly be obvious to a POSITA that—at this same time that a subset of the model is downloaded—a subset of the view-generating logic and controller logic could also be downloaded.

569. Clients downloading software from servers has been practiced for as long as it has been possible to do so. Since at least the mid 1990s, this was the most common method of software upgrade. Operating systems such as Microsoft Windows 95 (released in 1995) routinely used the ability of user machines (clients) to download updates upon request from Microsoft servers to keep those systems up-to-date.

570. Another example of clients downloading view-generating and controller logic associated with an application from the server would be Java applets, introduced in 1995 by Sun Microsystems. Java applets were small applications typically written in the Java programming language, and then compiled into what were called Java “bytecodes.” These bytecodes were then downloaded by a browser, and executed by an instance of the Java Virtual Machine (a separate program that ran in concert with the web browser). The user could launch a Java applet from a web page, or a Java applet could appear in a frame of the web page.

571. Applets were used to provide interactive features to web applications that could not be provided by HTML alone. They could capture keyboard and mouse input, which could then be used to update the displayed graphic content.

⁷²⁴ Roy; 4:51-55.

⁷²⁵ Roy; 5:50-64.



572. JavaScript, introduced in late 1995 by Netscape Navigator, and Jscript, introduced by Microsoft in 1996, are yet other examples of clients downloading view-generating and controller logic associated with an application from the server. As described in Section 6.G, JavaScript and Jscript allowed web developers to create web pages with client-side interactivity. Used in conjunction with HTML (HyperText Markup Language) and CSS (Cascading Style Sheets), JavaScript and Jscript provided the means to create dynamic webpage behavior (this combination was often referred to as Dynamic HTML, or DHTML).

573. In the 1990s, website developers routinely use JavaScript or Jscript to create code for client-side executable content. Web browsers of the time had a built-in JavaScript (or Jscript) engine that executed the downloaded code within the browser on the user's machine. This code, which typically included both view-generation and controller logic, was downloaded with its associated web page upon demand (when the website is visited within the browser).

574. These examples demonstrate that a POSITA in the relevant time frame would have a thorough understanding of the practice of downloading view-generating and controller logic from the server to the client upon demand. A POSITA wishing to implement a web-based application in the late 1990s would have routinely used this understanding in such an implementation.

575. Thus, a POSITA examining Roy would understand that the controller and view-generating logic could be downloaded to a client computer connecting to the game network. For example, a POSITA would understand that downloading all of the data and logic together would be more efficient than requiring multiple client/server interactions. Plus, it would be obvious to a POSITA to include the logic needed to process the data along with the data being sent. And in any case, the earliest point that data or logic could be transmitted would be after the client connected.



The only alternative would be if the player already had some of the data and/or logic on a hard disk (or previously installed from a hard disk). Indeed, a POSITA would understand that, at the relevant time, the only other option to deliver the data and logic to a user's computer would be via CD-ROM. Given what are essentially two options (sending the data and logic online or delivering it physically with a disk), a POSITA would be motivated to at least consider both options.

Obvious over Roy in light of Rover

576. To the extent Roy does not expressly disclose this limitation, it is further my opinion that it would have been obvious to a person of ordinary skill in the art at the time the '719 Patent was filed to modify the online multiplayer gaming system described in Roy based on Rover's teachings about moving application data and logic between a client and server on demand using relocatable dynamic objects ("RDOs").

577. A POSITA would have been motivated to modify the online multiplayer gaming system described by Roy based on Rover's teachings. For example, Roy and Rover are in the same field of endeavor (interactive distributed systems).

578. In particular, a POSITA examining Roy and the online multiplayer game system described therein would need to determine how to distribute game data and logic to each client computer. At the relevant time, a POSITA would understand that the logic could either be sent on a disk of some sort (e.g., CD-ROM) or sent over a network. A POSITA would be motivated to consider both options.

579. Rover teaches a method for downloading data and application logic over the network on demand. For example, Rover explains that "[t]he Rover toolkit combines relocatable dynamic objects and queued remote procedure calls to provide unique services for 'roving' mobile

applications.”⁷²⁶ “A relocatable dynamic object is an object with a well-defined interface that can be dynamically loaded into a client computer from a server computer (or vice versa) to reduce client-server communication requirements.”⁷²⁷ “The Rover toolkit also offers advantages for workstation applications by providing a uniform distributed object architecture for code shipping, object caching, and asynchronous object invocation.”⁷²⁸ “Depending on the power of the mobile host and the available bandwidth, Rover dynamically adapts and moves functionality between the client and the server.”⁷²⁹ For example, “[b]y migrating a GUI to the client, input events can be processed locally, even if the client becomes disconnected.”⁷³⁰ As another example, Rover describes an e-mail system “which is contained in RDOs retrieved from an HTTP server when the user first loads the browser”—i.e., on demand.⁷³¹

580. Rover further explains how to handle a situation wherein the client connection to a server is lost, which a POSITA would be motivated to incorporate into the online gaming system described by Roy to handle situations wherein one of the clients are disconnected. For example: “Queued remote procedure call is a communication system that permits applications to continue to make non-blocking remote procedure call requests even when a host is disconnected, with requests and responses being exchanged upon network reconnection.”⁷³²

581. A POSITA would be motivated to incorporate Rover’s methods for downloading a subset of logic over a network into the online gaming system described by Roy to download data and program logic from the server to each client device rather than, for example, require use of a

⁷²⁶ Rover; p. 156.

⁷²⁷ Rover; p. 156.

⁷²⁸ Rover; p. 156.

⁷²⁹ Rover; p. 158.

⁷³⁰ Rover; p. 159.

⁷³¹ Rover; p. 165.

⁷³² Rover; p. 156.

CD-ROM. A POSITA would also be motivated to utilize Rover's teachings concerning how to handle a client that becomes disconnected, as described above.

e. Roy Anticipates or Renders Obvious Claim 26 of the '719 Patent

[26P] A method for use in a client/server system of reducing interactions between a client and server in association with an application being accessed by the client at the server, the method comprising the steps of:

582. To the extent the preamble is considered to be limiting, in my opinion, Rover discloses or renders obvious a method for use in a client/server system of reducing interactions between a client and server in association with an application being accessed by the client at the server, for the reasons described above for claim preamble 1P and claim element 1C. I incorporate those discussions by reference here.

[26A] configuring the server to:

- (i) store a model associated with the application;**
- (ii) execute view-generating logic associated with the application; and**
- (iii) execute controller logic associated with the application; and**

583. In my opinion, Roy discloses configuring the server (e.g., computer 62 – when it has the master event server and master database) to store a model associated with the application (e.g., a database representing the world model for the game) and to execute view-generating and controller logic associated with the application, for the reasons described above for claim element 1A. I incorporate that discussion by reference here.

[26B] configuring the client to:

- (i) store at least a subset of the model associated with the application;**
- (ii) execute at least a subset of the view-generating logic associated with the application; and**
- (iii) execute at least a subset of the controller logic associated with the application;**

584. In my opinion, Roy discloses configuring the client (e.g., computer system 60) to store at least a subset of the model associated with the application (e.g., a subset of the database

[REDACTED]

comprising the world model for the online multiplayer application) and to execute at least a subset of the view-generating and controller logic associated with the application, for the reasons described above for claim 1B. I incorporate that discussion by reference here.

[26C] wherein one or more portions of the application are performed at the client without the client having to interact with the server,

585. In my opinion, Roy that one or more portions of the application are performed at the client without the client having to interact with the server, for the reasons explained above for limitation 1C. I incorporate that discussion by reference here.

[26D] and further wherein the client and the server both locally maintain at least a portion of the model and execute the view-generating and controller logic associated therewith;

586. In my opinion, Roy discloses that the client and the server both locally maintain at least a portion of the model and execute the view-generating and controller logic associated therewith, for the reasons explained above for limitations 1A, 1B, and 1D. I incorporate those discussions by reference here.

[26E] further wherein, in accordance with such a dual model-view-controller arrangement, a model comprises application data, rules, and algorithms affecting the data, a view comprises a screen or window representation of a subset of the model that the application chooses to display, and a controller comprises the logic that processes user requests, and causes the model to be changed and the view to be refreshed.

587. In my opinion, Roy discloses that, in accordance with a dual model-view-controller arrangement, a model comprises application data, rules, and algorithms affecting the data, a view comprises a screen or window representation of a subset of the model that the application chooses to display, and a controller comprises the logic that processes user requests, and causes the model to be changed and the view to be refreshed, for the reasons explained above for limitations 1A, 1B, and 1D. I incorporate those discussions by reference here.



588. I incorporate the discussions of limitations 1A, 1B, and 1D in their entirety here, but to summarize: Roy discloses a model that comprises application data, rules, and algorithms affecting the data (e.g., a database representing the world model for the game); Roy discloses a view that comprises a screen or window representation of a subset of the model that the application chooses to display (e.g., the view displayed on the monitors of the current virtual world); Roy discloses a controller comprising the logic that processes user requests, and causes the model to be changed and the view to be refreshed (e.g., logic that processes a user command and generates an “event”, which is then displayed).

7. Additional Opinions Relating to Dependent Claims 2 and 12

a. Asserted Claim 2 of the ’719 Patent Would be Obvious to a POSITA

589. In my opinion, the additional limitation recited in Asserted Claim 2 of the ’719 Patent (“The method of claim 1, wherein the client and server communicate over a HyperText Transport Protocol network.”) would be obvious to a person of ordinary skill in the art at the time the ’719 Patent was filed. Indeed, a POSITA from the relevant time would understand that *any* client/server system would (or could) utilize HTTP for communication. The bases of this opinion are set forth below.

590. According to Internet Live Stats, there were 17,087,182 websites in the year 2000.⁷³³ Virtually all of these websites were accessed using HTTP, since HTTP was the default Internet protocol used for this purpose. It is not an exaggeration to say that in the year 2000, accessing a website was synonymous with using HTTP. Any POSITA would have expected to use HTTP when developing a web-based application. No POSITA would have reasonably considered using HTTP in a web-based application as novel or innovative.

⁷³³ <https://www.internetlivestats.com/total-number-of-websites/>.



591. A POSITA would also have known and understood that the HTTP protocol was simply an application-layer protocol that employed the transport layer TCP protocol. A POSITA would have also known and understood the relationship between HTTP, TCP, and UDP, as described in Section 6.F. In 2000, TCP was routinely used for reliable in-order packet delivery on large area networks like the World Wide Web; UDP was used on small local area networks where performance goals outweighed reliability concerns. Making these kinds of choices was (and still is) routine practice in the design of distributed systems.

592. As I have already described, it is further my opinion that, to the extent that any of the prior art cited in this report is found not to disclose using HTTP in their client/server systems, a POSITA would have found it trivial to do so.

593. For example, Distributed Smalltalk employed UDP datagrams on the small local area network that connected machines in the Department of Computer Science at the University of Washington in 1987 (see Section 8.B.4). Reliable delivery of packets was not a concern in this case, since network traffic was light, and the number of machines limited. Had reliability been a concern, changing one line of code would have caused Distributed Smalltalk to use TCP instead of UDP (see Section 8.A.4).

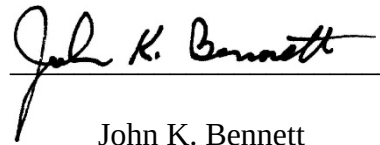
594. A POSITA would have routinely used HTTP to implement web-based applications in the late 1990s, and could have created web-based applications based on the teachings of any prior art reference cited in this report (e.g., the online multiplayer games described in Roy and Yelon, the online bookstore application described in Lowery, the Rover Applications described in Rover, the distributed applications described in Bennett, etc.). The fact that the networking components of these systems used TCP or UDP as the transport layer would have made the networking aspects of such an undertaking trivial, as the example in Section 8.A.4 demonstrates.

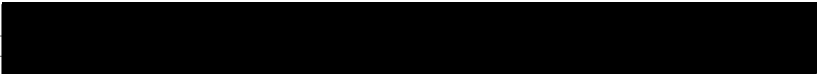
[REDACTED]

639. I understand that expert discovery is ongoing. I reserve the right to respond to any reports that are submitted by Plaintiff's expert witnesses or to any testimony by Plaintiff's fact or expert witnesses, whether at deposition or at trial. My opinions are subject to change based on additional information provided or positions taken by Plaintiff or its experts, or based on additional work I may perform. If appropriate, I may amend or supplement my opinions in a follow-on report.

640. I declare under penalty of perjury that the foregoing is true and correct to the best of my knowledge. If called upon to testify, I would do so consistent with the statements and opinions contained in this report, and in any follow-on reports.

Executed on February 2, 2024


John K. Bennett



**IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE**

INTERNATIONAL BUSINESS	§	
MACHINES CORPORATION	§	
	§	
Plaintiff	§	
	§	
v.	§	C.A. No. 22-590-GBW
ZYNGA INC.	§	
	§	
Defendant.	§	HIGHLY CONFIDENTIAL – ATTORNEYS’ EYES ONLY

**Expert Report of
John K. Bennett, Ph.D., P.E.
February 2, 2024**



Table of Contents

1. Engagement	4
2. Qualifications	4
3. Materials Reviewed	7
4. Summary of Expected Testimony	7
5. Understanding of the Relevant Law	9
A. Claim Construction	9
B. Anticipation	10
C. Obviousness	11
D. Written Description	13
E. Level of Skill in the Art	13
F. Non-Infringing Alternatives	14
6. Summary of Relevant Technical Background	15
A. Distributed Systems and Applications	15
B. Object-Oriented Languages and Systems	16
C. Distributed Object-Oriented Systems	17
D. Smalltalk-80	18
E. Model-View-Controller (MVC)	19
F. The Internet, HTTP and TCP/IP Network Communication Protocols	22
G. JavaScript, Dynamic HTML (DHTML), and the Document Object Model (DOM)	28
1. JavaScript	28
2. DHTML and DOM	29
7. Overview of the ' 719 Patent and Asserted Claims	30
A. Overview	30
B. Asserted ' 719 Patent Claims	37
C. Infringement Considerations	40
8. Bases of Opinions	40
A. Background: the " Teachings" of the ' 719 Patent Were Not Novel to a POSITA	4
1. Techniques for Reducing Network Interaction Were Known and Practiced	40
2. The MVC Paradigm Was Known and Practiced	42
3. The Dual MVC Approach Was Known and Practiced	42
4. HTTP was Known, Practiced, and Built Upon Known Lower-Level Protocols	43
B. The Asserted Claims of the ' 719 Patent are Anticipated or Rendered Obvious by the Prior Art	44

1.	Lowery Anticipates or Renders Obvious Asserted Claims 1, 2, 12, and 26 of the ' 719 Patent	45
2.	Rover Anticipates or Renders Obvious Asserted Claims 1, 2, 12, and 26 of the ' Patent	719 86
3.	Hyndman Anticipates or Renders Obvious Asserted Claims 1, 2, 12, and 26 of the ' 719 Patent	130
4.	Bennett Anticipates or Renders Obvious Asserted Claims 1, 2, 12, and 26 of the ' 719 Patent	175
5.	Yelon Anticipates or Renders Obvious Asserted Claims 1, 2, 12, and 26 of the ' Patent	719 207
6.	Roy Anticipates or Renders Obvious Asserted Claims 1, 2, 12, and 26 of the ' Patent	719 244
7.	Additional Opinions Relating to Dependent Claims 2 and 12	296
8.	No Secondary Considerations Render the Asserted Claims Nonobvious	299
9.	The Asserted Claims May Lack Written Description, Depending on Plaintiff's Infringement Argument	300
10.	Non-Infringing Alternatives	303
A.	Non-Infringing Alternative No. 1	303
B.	Non-Infringing Alternative No. 2	305
11.	Concluding Remarks	307



1. Engagement

1. In September 2023, I was retained by counsel to Zynga to evaluate certain aspects of U.S. Patent 7,702,719 B1 (the ' 719 Patent), and to offer my own analysis and opinions regarding matters related to that Patent. I understand that Plaintiffs are asserting only Claims 1, 2, 12, and 26 of the ' 719 Patent (the Asserted Claims); I have considered only those claims in the preparation of this report. In my reading and understanding of the ' 719 Patent, I have been guided by the Court's Construction of Claim Terms for that Patent.¹

2. I am prepared to be called as an expert witness if my testimony is requested in this matter. This report contains a summary of my analysis to date. I am continuing my work on the case, and I may supplement, refine, or revise my opinions and findings as a result of further review and analysis. I am being paid \$800 per hour for work in this matter. I have no business relationships with Zynga or IBM.

2. Qualifications

3. I am a Professor Emeritus of Computer Science and Electrical and Computer Engineering at the University of Colorado at Boulder. Over my career, my research has focused broadly in the area of operating system/microprocessor architecture interaction and, more narrowly, in the area of distributed and parallel computing systems. My teaching has included graduate and undergraduate courses in computer architecture, operating system design, distributed systems, computer system design, and other topics.

4. My titles at the University of Colorado included Director of Inworks, Associate Vice Chancellor for Innovation Initiatives, and Professor of Computer Science at the University of Colorado Denver | Anschutz Medical Campus; and Archuleta Professor of Computer Science

¹ *IBM v. Zynga Inc. et al.*, No. 1:22-cv-00590-GBW (D. Del. October 30, 2023), ECF No. 171.

[REDACTED]

and Engineering and Professor of Electrical, Computer and Energy Engineering at the University of Colorado at Boulder. I served as the Director of the ATLAS Institute and Associate Dean of Engineering for Education in the College of Engineering and Applied Sciences at the University of Colorado at Boulder, and as the Master of Richardson College at Rice University.

5. I have a doctorate in Computer Science from the University of Washington. My doctoral dissertation involved the design, implementation and evaluation of a Smalltalk system distributed over a local area network. Smalltalk is an object-oriented programming environment developed at the Xerox Palo Alto Research Center that was a precursor of Oracle's Java technology. I have done pioneering work in both distributed shared memory systems and adaptive software techniques for shared memory parallel computing. Prior to completing my thesis work at the University of Washington, I was President of Pacific Mountain Research, Inc. (PMR) in Seattle, Washington.

6. For the past forty-five years, I have focused on the development of computer software and hardware, both directly as a creator of sophisticated software applications and operating systems, and as an academic, studying and contributing to the field. Examples of commercial and academic computer systems that I have helped develop include the mission control system for Jet Propulsion Laboratories, the control system for the Washington D.C. Metropolitan Transit Authority light rail system, the propulsion control system for the Spruance class gas-turbine-powered destroyer, the Eden distributed computer system, an object-oriented distributed computing system based upon Smalltalk, several multiprocessor and virtual memory systems, several high performance graphics and image processing systems, and several very large scale integrated (VLSI) circuits, biomedical instrumentation systems, and video processing systems. These systems were utilized or marketed by several companies, including Motorola, Signetics,

[REDACTED]

RCA, IBM, Eastman Kodak, Microsoft, Sequent, Texas Instruments, Weyerhaeuser, the Department of Defense, and the New York Stock Exchange.

7. I have published over eighty research papers, and have given presentations at numerous conferences, universities, and research laboratories. A list of these papers can be found in Appendix A.

8. Over the years, I have received significant research funding from industry and government organizations in support of my work in over thirty-one research and educational projects, including support from the National Science Foundation, the Defense Advanced Research Project Agency, NASA, the National Institute of Health, the National Institute of Disability and Rehabilitation Research, and the Texas Higher Education Coordinating Board, as well as grants from Compaq, Intel, Tandem, Microsoft, Ansell, Regent, Bell Northern Research, Hewlett-Packard, and IBM. (A listing of my grants for the past fifteen years is attached as part of Appendix A.)

9. I have been an active member of the Association for Computing Machinery since 1979, and the Institute for Electrical and Electronic Engineers (in which I hold the rank of Senior Member) since 1976. I am a registered Professional Engineer (inactive) in the State of Texas. I am the past president of the national Governing Board of Engineers Without Borders USA, and have received several teaching awards, including the W.M. Keck Foundation National Award for Engineering Teaching Excellence. I am a named inventor on two U.S. Patents.

10. As a result of this experience, I am thoroughly familiar with distributed and object-oriented system design and development, hardware and software development, and software testing and support. I have participated in the development and testing of commercial software products as both a software developer and manager. In the course of my research and teaching, I

[REDACTED]

have also become broadly familiar with various client and server operating systems, and a variety of applications development and enterprise programming environments.

11. Over the past thirty plus years, I have been qualified as an expert in computer systems in courts throughout the United States and Canada, and have given testimony on subjects relating to computer software in more than a dozen cases. My expert testimony has included analysis of computer software behavior, comparisons of functionally similar software and hardware, patent claim interpretation and patent claim analysis. A listing of the cases in which I have testified is contained in Appendix B.

3. Materials Reviewed

12. My knowledge in these areas (and, therefore, my opinions) derives in part from my active involvement in computer systems research and development throughout my career. As part of my teaching and academic advancement, I have kept current on issues in computer software, including (specifically) operating and distributed system development.

13. Aside from my background in computer science, in the course of preparing this report I have taken into consideration the following materials:

- a) the various complaints, motions, briefs, interrogatory responses and other pleadings submitted by the parties in this matter;
- b) publicly available documents relating to issues in the case, including patents, articles, books, and websites; and
- c) other documents and information specifically cited in this report.

4. Summary of Expected Testimony

14. I generally expect to testify on the following subjects:

- a. The level of knowledge that would be expected of a person of ordinary skill in the art (POSITA) during the time frame relevant to the ' 719 Patent.

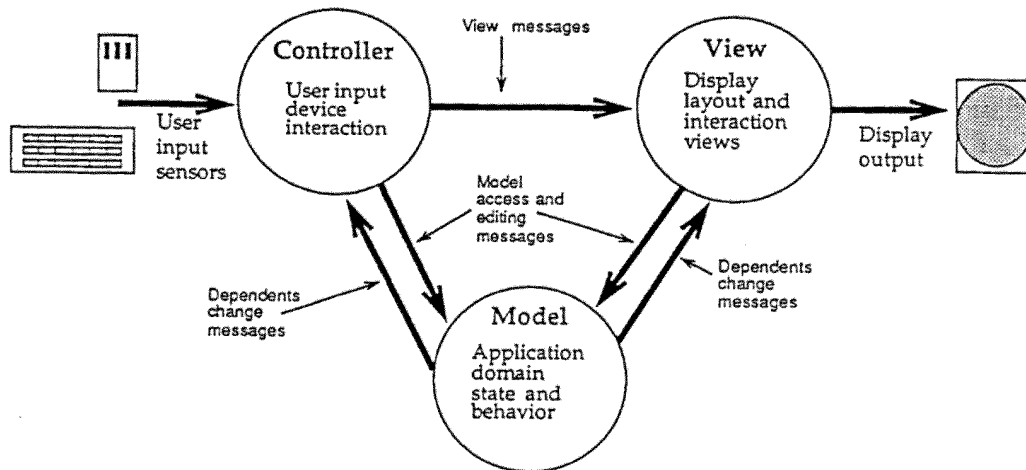
Second, the reactive principle requires that *every component accessible to the user should be able to present itself in a meaningful way for observation and manipulation.*

61. Of particular relevance to this case, these two principles led to the Model-View-Controller (MVC) – based graphical user interface design that was an essential component of Smalltalk-80.

E. Model-View-Controller (MVC)

62. Model-View-Controller (MVC) is an object-oriented user interface design methodology, or *metaphor*, that divides the user interface into three related components: Model, View, and Controller. It was created during the development of Smalltalk-80. As implemented in Smalltalk-80, objects of three classes made up the MVC: the *Model*, which maintained the state of an object, the *View*, which creates a representation of that state for viewing, and the *Controller*, which managed user interaction between the Model and the View (i.e., updates to the object's state). A Model can be simple, e.g., an integer counter, or complex, e.g., a database. Controllers communicate with the Model, and provide the interface between the Model, its associated Views (of which there may be several), and the user interface. Applications in Smalltalk-80 usually, but not always, had a single Model, and several view/controller pairs. Views request data from their model, and create representations of objects' data that they receive for subsequent display. These relationships are shown in the figure below.¹²

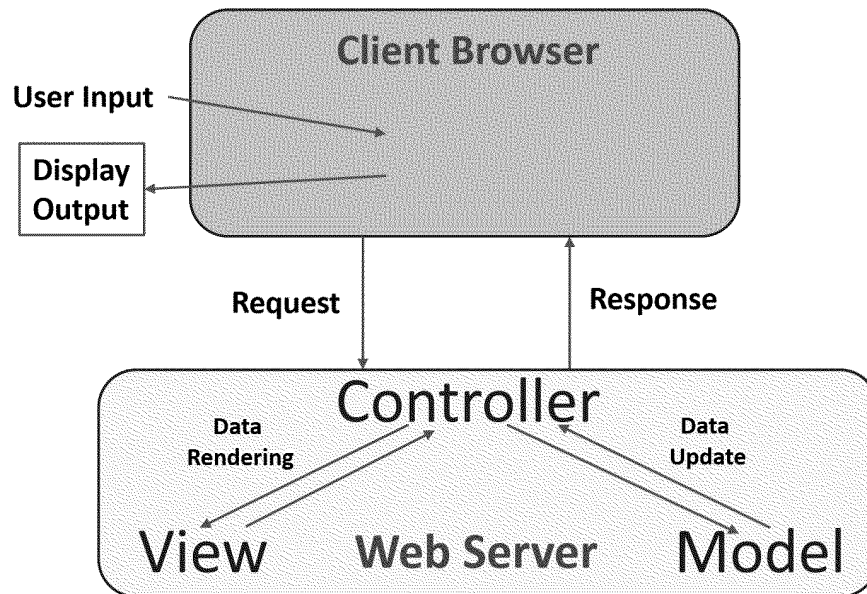
¹² G. Krasner & S. Pope, *A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System*, Journal of Object-Oriented Programming, Vol. 1, p. 5, Jan. 1988.



63. Controllers typically maintain the interface between their associated Models and Views and the user interface devices (keyboard, pointing device, etc.). Put simply, a Model is a collection of data (e.g., a list of library books); a View creates representations of this data (e.g., sorting the books by author, title, or subject); and a Controller provides mechanisms for manipulating the model (e.g., adding or checking out a book). In this approach, Views and Controllers have exactly one Model, but a Model can be associated with several Views and Controllers. As shown in the figure above, Smalltalk-80 display output is managed by a View. Other MVC systems place this responsibility with a Controller.

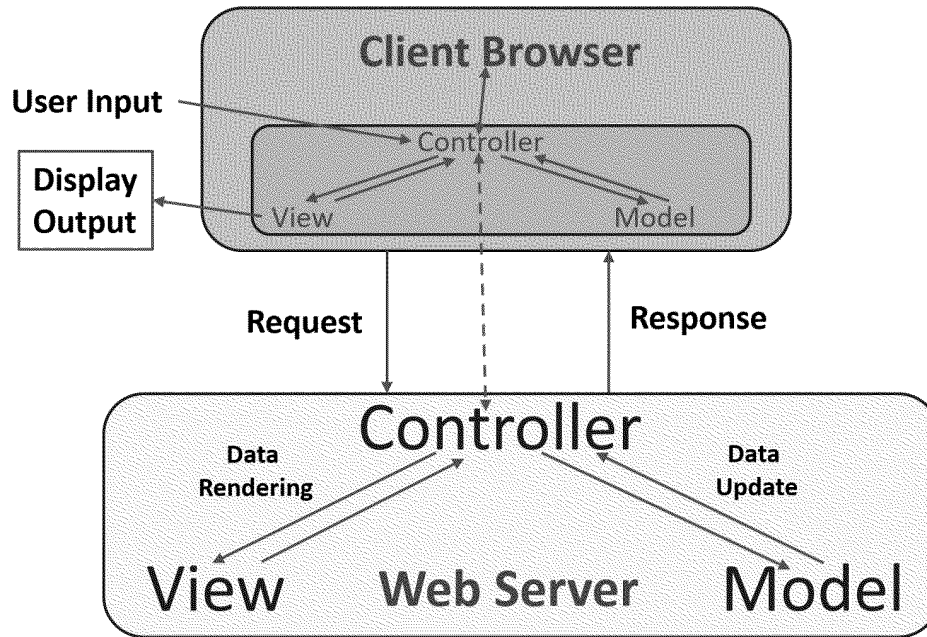
64. Although MVC was developed for desktop graphical user interface, its simplicity, encapsulation, and modularity made it popular for designing early web applications. This usage increased after NeXT incorporated MVC into WebObjects (first released in 1996). When using MVC for web application design, the client browser typically becomes another just user interface mechanism (and a gateway for other user interactions), as depicted in the figure below.

Standard MVC Web Application



65. As mechanisms for embedding code within MVC-based browsers (e.g., JavaScript) were developed in the 1990s, it became natural to begin dividing MVC responsibilities between server and client. Although dual MVC distributed applications had been built prior to MVC use in web application development, e.g., Distributed Smalltalk, dual MVC became more widespread with the advent of web application software environments that facilitated such development. The figure below depicts a common configuration of the dual MVC model employed in a web application. In this example, portions of the application Model, View and Controller reside on the client. Typically, these portions are cached copies of some part of their associated component located on the server. Web applications communicate using HTTP at the application level. The dashed green line depicts “private” Controller to Controller communication that may be embedded into the HTTP communication.

Dual MVC Web Application



66. MVC evolved over time, giving rise to several variants, including Hierarchical Model– View– Controller (HMVC), Presentation– Abstraction– Control (PAC), Model– Adapter (MVA), Model– View– Presenter (MVP), Model– View– View-Model (MVVM), and others.

F. The Internet, HTTP and TCP/IP Network Communication Protocols

67. The Internet has a storied history dating back to the early 1960s. The technologies that underly the Internet: a network of networks of computers, packet switching, and hypertext had their roots in the groundbreaking work of Kleinrock (packet switching on digital networks), Licklider (“ Intergalactic Computer Networks”), Baran (store and forward message routing), and Nelson and van Dam (hypertext).

68. The communication protocols used to send and receive network messages became standardized when Cerf and Kahn published “ A Protocol for Packet Network

82. DHTML and the DOM allowed website developers to add effects that were otherwise difficult to achieve, simply by changing the Document Object Model (DOM) and page style. This combination of HTML, CSS, and JavaScript offered ways to:

- animate text and images in web documents,
- embed dynamic content (e.g., a counter) that automatically refreshed its content without communication with the server,
- capture user input (such as pressing a key and clicking the mouse), and then process, verify and respond to that input without having to send data back to the server, and
- incorporate rollover buttons and drop-down menus into web pages.

83. These website development capabilities, which were routinely available at the time the ' 719 Patent was filed, appear to encompass the claimed inventions of the ' 719 Patent.

7. Overview of the ' 719 Patent and Asserted Claims

A. Overview

84. The ' 719 Patent describes a dual MVC-based web application intended to be employed in a client/server distributed system. In particular, the ' 719 Patent seeks to both improve application performance and ease software development and maintenance efforts:

“ The invention relates to web applications and, more particularly, to an architecture and method of programming which improves performance and eases the development and maintenance effort. The invention addresses performance by employing a dual-MVC approach, in which a subset of the application's Model-View-Controller reside on the client, and the full Model-View-Controller and View-Generating Logic reside on the server, thereby reducing the number of required server interactions. In contrast to the fat-client architecture, the invention does not require any application installation or persistence on the client.”²⁴

²⁴ ' 719 Patent; 2:39-49.

85. The described performance improvement is attributed in part to the application downloading small portions of code that contain MVC software. This software is downloaded when needed (“ upon demand”) and executed without the need to permanently install the software:

“ The dual-MVC approach of the invention avoids the installation and disk-space problems because client-side code associated with the MVC of the application is preferably downloaded from the server on demand by the user of the client device, avoiding the install process, and is preferably not saved permanently to the disk of the client.”²⁵

86. Thus, at various times during the execution of the application, portions of the application’ s model, view, and controller reside on the client. The server always contains the authoritative version of the application’ s model, view and controller:

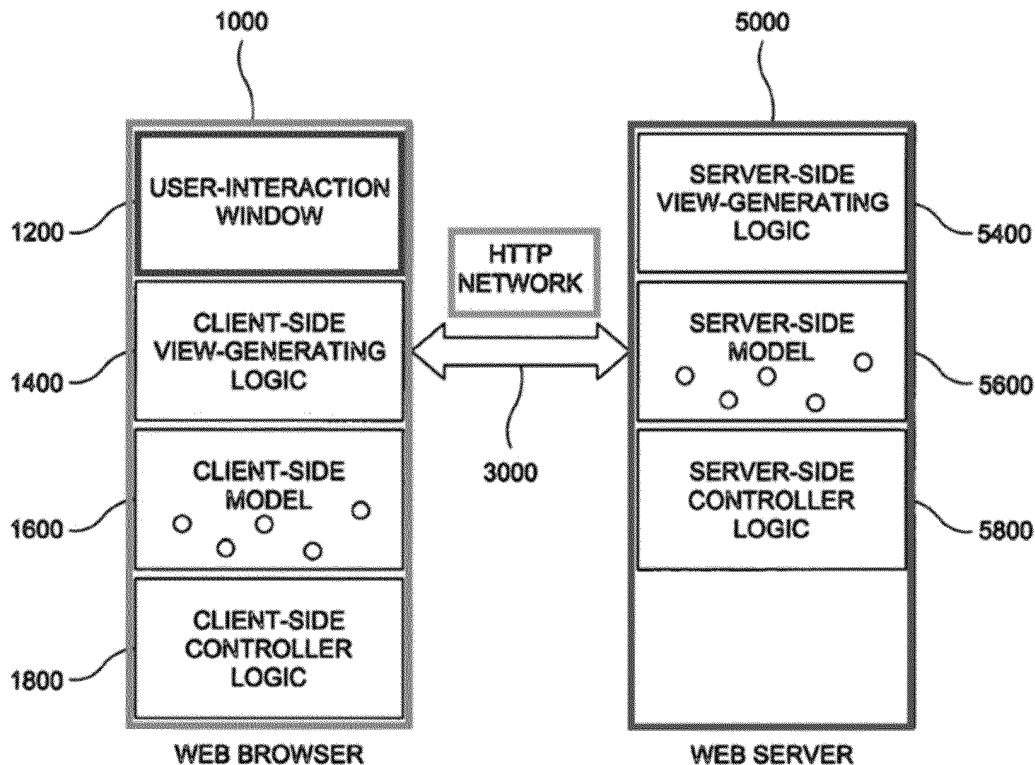
“ The invention addresses performance by employing a dual-MVC approach, in which a subset of the application’ s Model-View-Controller reside on the client, and the full Model-View-Controller and View-Generating Logic reside on the server, thereby reducing the number of required server interactions.”²⁶

87. This idea of the model, view, and controller being located on both the client and the server is depicted in the block diagram of the “ dual-MVC (Model-View-Controller) approach” shown below.²⁷

²⁵ , 719 Patent; 3:1-6.

²⁶ , 719 Patent; 2:42-47.

²⁷ , 719 Patent; Figure 3 (coloration added).



88. In this figure, the “User Interaction Window 1200” is a web browser. All other boxes are the individual components of the Model-View-Controller found on both client (surrounded by a blue box) and server (surrounded by a green box). The “HTTP Network 3000” is presumably a standard network interface capable of supporting HTTP, although the version of HTTP to which the patent’s authors refer is not specified. The server-side MVC components (5000) are considered to be the “authoritative” (complete) copies of these components; the client-side versions of these components (1000) are subsets of the server version (“a subset of the application’s Model-View-Controller reside on the client, and the full Model-View-Controller and View-Generating Logic reside on the server”²⁸).

²⁸ 719 Patent; 2:43-46.

6. **Roy⁶¹⁸ Anticipates or Renders Obvious Asserted Claims 1, 2, 12, and 26 of the ' 719 Patent**

477. U.S. Patent 5,838,909 (Roy) is entitled “ Reducing Latency When Synchronizing Access to a Multi-User Database Over a Network.”⁶¹⁹ Roy was filed May 23, 1996, and issued Nov. 17, 1998. Because its filing date precedes that of the ' 719 Patent, I understand that Roy meets the legal standards required to qualify as prior art to the ' 719 Patent. Roy is neither identified on the face of the ' 719 Patent, nor was it considered during prosecution.

478. In my opinion, Roy anticipates or renders obvious all the limitations of Asserted Claims 1, 2, 12, and 26 of the ' 719 Patent. In the alternative, to the extent any limitation is found missing from Roy, Roy renders all the Asserted Claims of the ' 719 Patent obvious in light of the knowledge possessed by one of ordinary skill in the art and/or the specific prior art references I discuss below. The bases for this opinion are set forth below.

479. Note: In some instances, Roy uses the terms “ Master” and “ Slave” to r components of the described invention. These terms, which trivialize the horrific practice of treating people as property, are not part of the modern engineering lexicon. Since these terms are essential to being able to describe the patent in the language in which it was written, I will, where necessary, also use them here.

a. A Brief Description of U.S. Patent 5,838,909 (Roy)

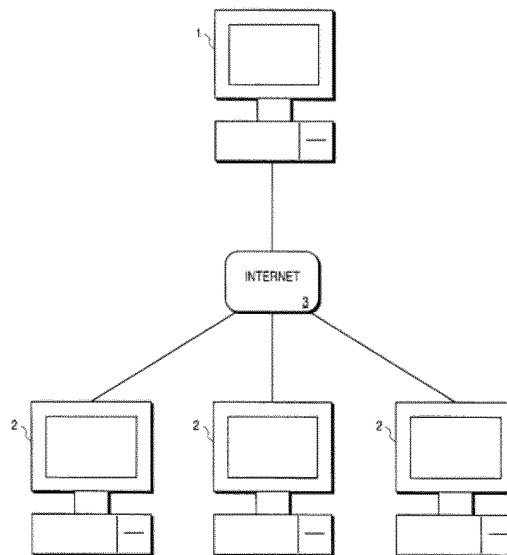
480. Roy generally discloses a distributed software system targeted toward multi-player games on the Internet. As shown in the figure below, Roy discloses a “ central server computer 1” coupled via the “ Internet 3” to a number of “ client computer⁶²⁰As disclosed by Roy,

⁶¹⁸ U.S. Patent 5,838,909 (“ Roy”).

⁶¹⁹ Roy; Cover.

⁶²⁰ Roy; 3:48-51 and Figure 1.

both the server and all clients execute the game software referred to in Roy as an “event server,” as further described below.



481. Each node (i.e., computer or server) in the system contains a *database* and an *event server*. The database at each node contains a model of a fictitious “game world.” The event server at each node is responsible for processing local user inputs (“events”) that update the game world database, as well as synchronizing with other nodes’ event servers (via the “master event server” described below), so that their copies of the database are kept up to date.⁶²¹

482. As disclosed in Roy, one database is designated as the “master database,” and one event server is designated as the “master event server.” These components are typically located on the same node. The “master database” is considered to be the authoritative copy of game state. Roy teaches that the node so designated should be the node whose user is most active (i.e., the one with the most restrictive latency requirements). The idea is that by placing the “master” components near the most active user, in-game latency will be reduced.⁶²²

⁶²¹ Roy; 1:66-2:10.

⁶²² Roy; 3:16-47.

483. Since the most active user is likely to change during game play, Roy also teaches a technique for migrating the master event server to another node (the one whose user has become most active). The figure below depicts these relationships.⁶²³

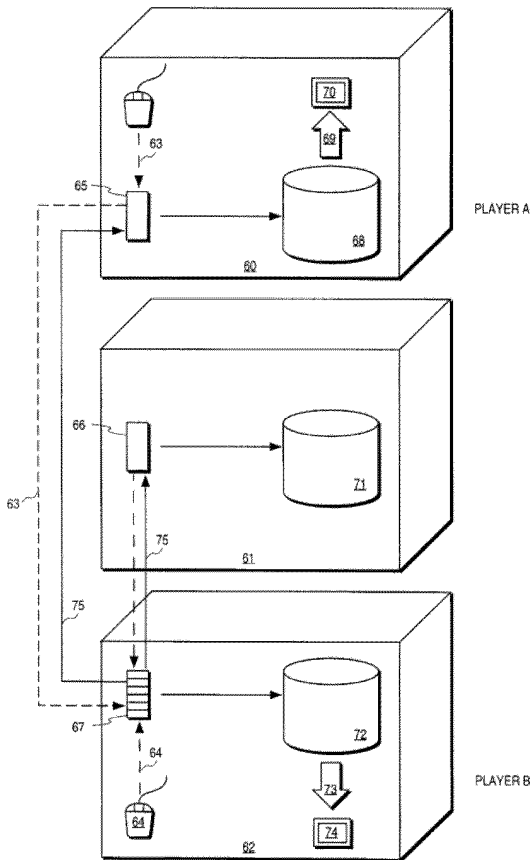


FIG. 6

484. Slave copies of the master database (or portions thereof) are stored on each machine other than the machine in which the master event server is located. By providing one master copy of the database, events originating from different machines can be more easily synchronized. By providing slave copies of the database in every other machine involved in the game, the need to send rendering commands across the network is reduced or eliminated, which reduces bandwidth consumption.⁶²⁴ An event server generally performs four primary functions:

⁶²³ Roy; Figure 6 and 8:60-9:5.

⁶²⁴ Roy; 4:47-54.

1. connecting to other event servers,
2. routing events to the local database and to other event servers (including the master event server),
3. migrating the master event server when appropriate, and
4. reconnecting to a new event server such that game events are not lost or duplicated.⁶²⁵

485. The figure below depicts an implementation of a multi-user game played over the Internet as disclosed by Roy.⁶²⁶

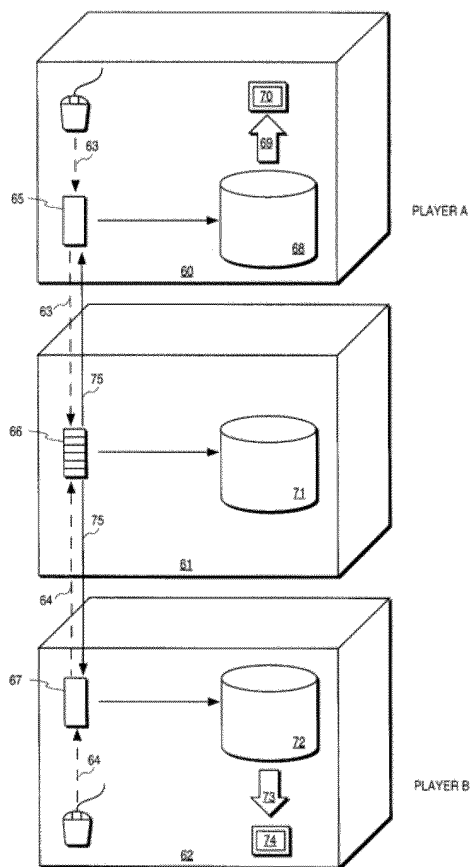


FIG. 5A

486. This figure depicts a game played by two players, PLAYER A and PLAYER B. PLAYER A is the user operating computer system 60; PLAYER B is the user operating computer

⁶²⁵ Roy; 4:62-5:2.

⁶²⁶ Roy; Figure 5A.



system 62. Initially, the master database 71 and the master event server 66 are located in a central server computer system 61. Also shown within computer system 60 (used by Player A) are a slave event queue 65 and a slave copy 68 of master database 71. Similarly, located in computer system 62 (used by player B) is a slave event server 67 and a slave copy 72 of master database 71.⁶²⁷

487. The display 74 shown within computer system 62 is updated by rendering commands 73 from the slave copy 72 of master database 71, rather than from the master database 71. This design choice is intended to reduce bandwidth consumption on the network.⁶²⁸

488. In the same manner, the display 70 shown within computer system 60 is updated by rendering commands 69 from slave copy 68 of master database 71, rather than directly from master database 71.⁶²⁹

489. User events 64 (such as keyboard presses or mouse movement or button clicks) from computer system 62 are received by slave event server 67 and then passed along to the master event server 66, unless these events are purely local (events that are only local to computer system 62). In the same manner, events 63 originating from computer system 60 are received first by slave event server 65 and then passed along to master event server 66, unless those events are purely local to computer system 60. Events received by the master event server 66 are used to update the master database 71. These events are also passed along to all slave event servers as master events 75.⁶³⁰

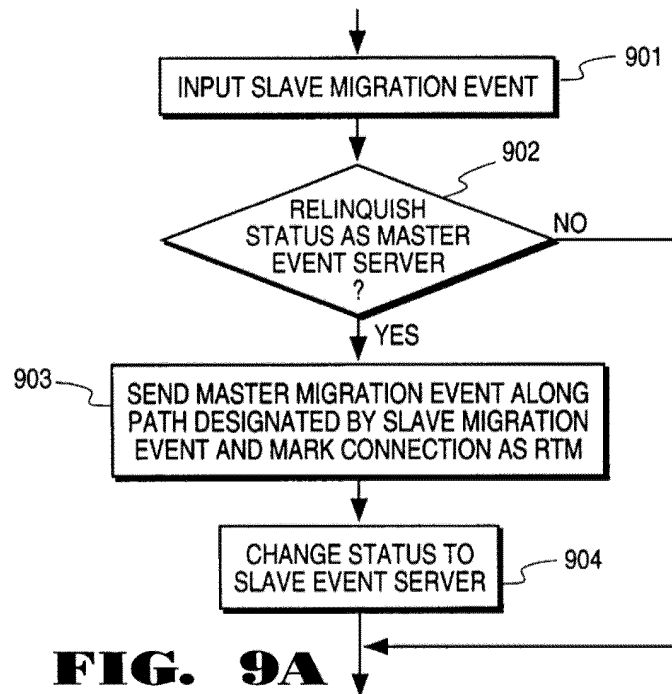
⁶²⁷ Roy; Figure 5A and 5:17-29.

⁶²⁸ Roy; Figure 5A and 5:29-33.

⁶²⁹ Roy; Figure 5A and 5:33-36.

⁶³⁰ Roy; Figure 5A and 5:37-49.

490. When a determination is made of which computer system participating in the game is used by the player having the role that requires the least latency, the master event server 66 is migrated to the machine used by that player, as depicted in the figure below.⁶³¹



491. This figure depicts the process of migration from the viewpoint of the master event server. The process begins when the current master event server receives a request in the form of a slave migration event (step 901). This request would typically come from a client that has determined that its user requires lower latency (i.e., this user is very active in game). The master event server then makes a determination (using game logic criteria not relevant to this discussion) whether to relinquish its status as the master event server (step 902). If the decision is made to relinquish its status as the master event server, then the master event server sends a “master migration event” along the path by which it received the slave migration event, which will ultimately reach the requesting client. All computers along that path will mark the network path to

⁶³¹ Roy; Figure 9A and 5:65-6:2.

the requesting client as the Route to Master (RTM) (step 903). The master event server will then change its status to slave event server (step 904).⁶³²

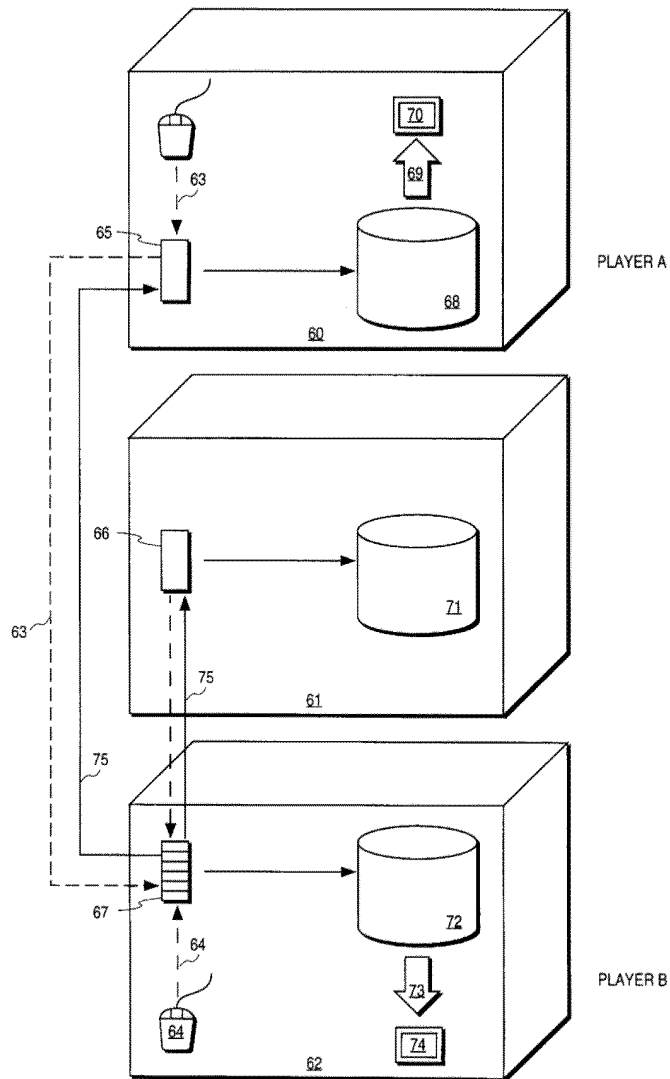


FIG. 6

492. More internal details can be seen in the figure above, where each box (60, 61, and 62) represent nodes. The cylinders 68, 71, and 72 represent copies of the local database within these nodes that contain the game state. Boxes 70 and 74 represent the displays on these nodes.

⁶³² Roy; Figure 9A and 7:60-8:5.

Three-button mice (63 and 64) generate user input events, which are handled by event servers 65 and 67, respectively. Event server 67 (with small horizontal lines) is the “master event server”⁶³³.

493. Thus, we can relate the language of Roy to that of the ’ 719 Patent, as follows: the Roy “master database” and “master event server” constitute the “server” as described in the ’ 719 Patent; and the Roy “slave database” and “slave event server” constitute the “client” in the ’ 719 Patent.

494. Roy also teaches an embodiment in which the master database is divided into multiple sections, or “objects”, each of which may reside on a different machine. Each of these disparate sections (located on different nodes) functions as a “master database” relating to a particular aspect of the game. A separate “master event server” is associated with each of these sections.⁶³⁴

b. Roy Anticipates or Renders Obvious all Elements of Claim 1 of the ’ 719 Patent

[1P] A method for use in a client/server system of reducing interactions between a client and server in association with an application being accessed by the client at the server, the method comprising the steps of:

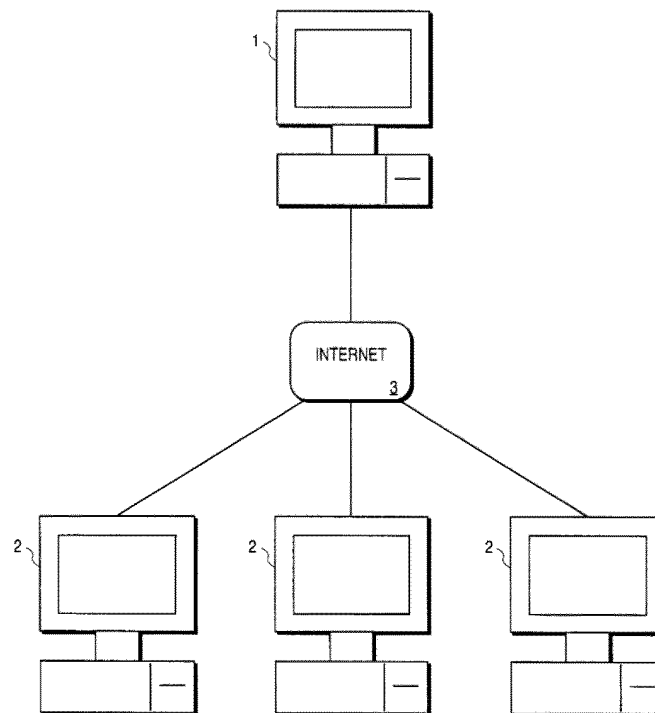
495. To the extent the preamble is limiting, in my opinion, Roy discloses a method for use in a client/server system (e.g., the client/server systems depicted in Figures 1, 4, 5, and 6) of reducing interactions between a client and server (e.g., Roy discloses that interactions between client computer 60 and server computer 61 when client computer 60 processes a “local event” do not require any interaction with the central server 61) in association with an application being accessed by the client at the server (e.g., a user connects to an online multiplayer game).

⁶³³ Roy; Figure 6 and 8:60-9:5.

⁶³⁴ Roy; 4:56-61.

496. First, Roy discloses a client/server system. Roy explains the operation of the claimed system as follows:

“ Referring now to FIG. 1, a network configuration in which the present invention may be implemented is illustrated. A central server computer 1 is coupled via the Internet 3 to a number of client computer systems 2. The server computer system 1 and each of the client computer systems 2 includes stored game software, including software in which the present invention is embodied. The game software includes various application program interfaces (APIs) providing an interface to operating system software.”⁶³⁵



“ A method is provided of reducing network latency in accessing a database in a network. The network has a number of nodes, including a first node and a second node. The first node includes a first event server for sequencing commands to the database, which is stored in one of the nodes in the network. The second node includes a second event server for sequencing commands to the database. The method comprises the steps of changing the status of the first event server from master event server to slave event server based on a network latency criterion, and assigning the status of master event server to the second event server based on the network latency criterion.”⁶³⁶

“ FIG. 4B illustrates a multi-player game running in a network using a client-server configuration.”⁶³⁷

⁶³⁵ Roy; Figure 1 and 3:48-56 (highlighting added).

⁶³⁶ Roy; 1:66-2:10 (highlighting added).

⁶³⁷ Roy; 2:31-32 (highlighting added).

FIG. 4B shows a well-known configuration for implementing a multi-user game on a network. In the configuration of FIG. 4B, the game world is provided by a database 53 located on a central server 51. Client computer systems 50 and 52 are used by Players A and B of the game, respectively. Software 54 located in the central server 51 receives events 58 from computers 50 and 52 and provides these events to the database 53 in order to update the game world. Rendering commands 56 are then sent from the updated database 53 in the central server 51 across the network to update the displays 55 and 57 of client computers 50 and 52, respectively. As mentioned above, implementing as shown in FIG. 4B on the Internet can be problematic if either of players A and B are subject to a significant latency restriction. In addition, the transmission of rendering commands 56 from server 51 to clients 50 and 52 can be hindered by bandwidth limitations of the network. Consequently, the present invention provides a technique for overcoming such difficulties. In the technique of the present invention, every machine that participates in the game runs software referred to herein as an 'event server.' One of the event servers is known as the 'master event server' and is responsible for sequencing all changes to a master database representing the game world. All event servers other than the master event server are considered to be 'slave event servers'. In the preferred embodiment, the master database is located on the same machine as the master event server. Slave copies of the master database (or portions thereof) are stored on each machine other than the machine in which the master event server is located. By providing one master copy of the database, events originating from different machines can be more easily synchronized. By providing slave copies of the database in every other machine involved in the game, the need to send rendering commands across the network is reduced or eliminated, which reduces bandwidth consumption.”⁶³⁸

497. Second, Roy discloses a method for reducing client/server interaction in association with an application being accessed by the client. For example, Roy explains how the need to send rendering commands across the network is reduced by providing slave copies of the database to every computer connected to the game, and how, for example, local events are processed locally (i.e., without server interaction) by a respective gamer's computer, and the results are then displayed on that player's computer:

“By providing one master copy of the database, events originating from different machines can be more easily synchronized. By providing slave copies of the database in every other machine involved in the game, the need to send rendering commands across the network is reduced or eliminated, which reduces bandwidth consumption.”⁶³⁹

⁶³⁸ Roy; 4:18-54 (highlighting added).

⁶³⁹ Roy; 4:48-54 (highlighting added).

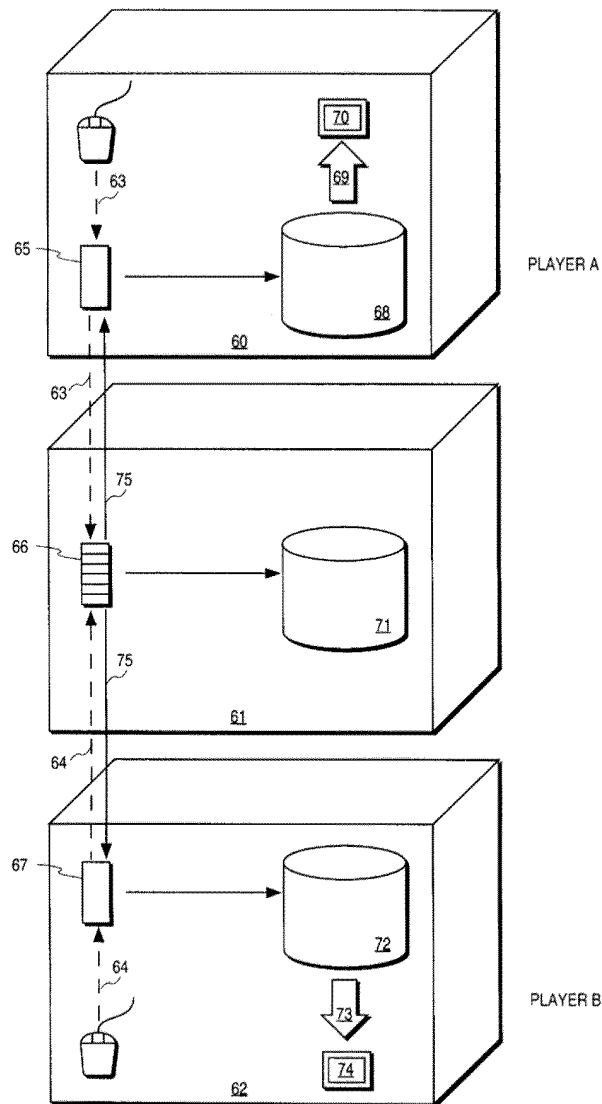


FIG. 5A

FIG. 5A illustrates a technique according to the present invention for implementing a multi-user game over the Internet. In FIG. 5A, the game is played by two players, players A and B, operating computer systems 60 and 62, respectively. It is noted, however, that the present invention is equally applicable to a game played simultaneously by more than two players. Initially, the master database 71 and the master event server 66 are located in a central server computer system 61. Located in computer system 60 (used by player A) are a slave event queue 65 and a slave copy 68 of master database 71. Similarly, located in computer system 62 (used by player B) is a slave event server 67 and a slave copy 72 of master database 71. The display 74 of computer system 62 is updated by rendering commands 73 from the slave copy 72 of master database 71, rather than from the master database 71 itself, thereby reducing bandwidth consumption on the network. Similarly, the display 70 associated with computer system 60 is updated by rendering commands 69 from slave copy 68 of master database 71, rather than directly from master database 71. Events 64 from computer system 62 are received

[REDACTED]

by slave event server 67 and then passed along to the master event server 66, unless the events 64 are purely local events for updating only the slave copy 72 of the database. Purely local events might be, for example, events which set display options or other functions that are only relevant to computer system 62. Similarly, events 63 originating from computer system 60 are received first by slave event server 65 and then passed along to master event server 66, unless those events are purely local to computer system 60. Events received by the master event server 66 are used to update the master database 71 and are passed along to all slave event servers as master events 75.”⁶⁴⁰

498. For example, as described, “ local events” are processed on the client and therefore do not require any server interaction.⁶⁴¹ Accordingly, interactions between the client and server are reduced. In particular, Roy discloses that an “ event” is created each time a user issues a command. If the event is determined to be a local event, then it is processed by the client and the results displayed.⁶⁴² As noted above, examples of local events include “ events which set display options or other functions that are only relevant to [that] computer system.”⁶⁴³

499. As another example, Rover teaches that, if a player’s computer becomes an effective server (i.e., the master event server and master database are migrated to that computer), then all events generated by that player are processed locally, regardless of whether they are local events or master events (i.e., interactions are reduced with the other computers including central server 61).⁶⁴⁴ Thus, for example, if the master event server and master database were migrated from central server 61 to computer system 62, all events generated by player B on computer system 62 would be processed by computer system 62, thus reducing interactions with central server 61 and computer system 60. This is depicted by Figure 6:

⁶⁴⁰ Roy Figure 5A and 5:17-49 (highlighting added).

⁶⁴¹ Roy; Figure 5A.

⁶⁴² Roy; Figure 5A.

⁶⁴³ Roy; 5:40-43.

⁶⁴⁴ Roy; 6:5-14.

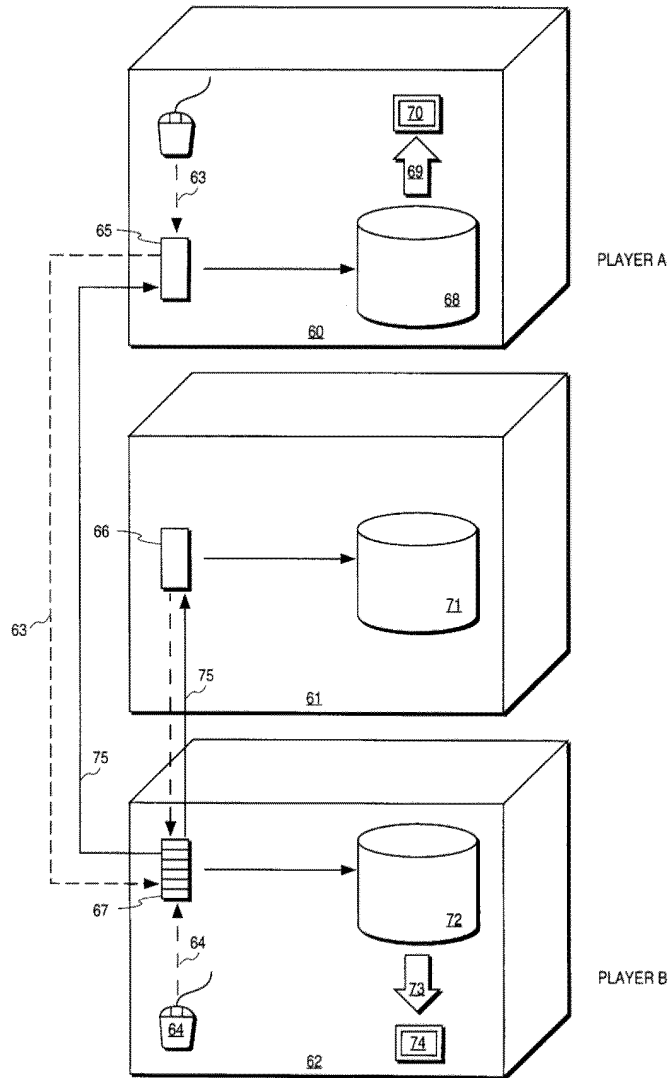


FIG. 6

500. Third, Roy discloses an application (e.g., an online multiplayer game) being accessed by the client (e.g., computer systems 62 or 60) at the server (e.g., central server 61 and/or whichever computer system has the master event server and master database).

501. For example, Figure 6 depicts how client computers 60 and 62 access an online application at central server 61. Or, as Roy describes, “ FIG. 6 illustrates a multi-player game

running in a network in which a slave event server has established a direct connection to the master event server.”⁶⁴⁵

502. As another example, Roy further discloses how the client computers and server computer transmit/receive event data to/from one another (i.e., how the client accesses the online multiplayer game application at the server):

“ A method is provided of reducing network latency in accessing a database in a network. The network has a number of nodes, including a first node and a second node. The first node includes a first event server for sequencing commands to the database, which is stored in one of the nodes in the network. The second node includes a second event server for sequencing commands to the database. The method comprises the steps of changing the status of the first event server from master event server to slave event server based on a network latency criterion, and assigning the status of master event server to the second event server based on the network latency criterion.”⁶⁴⁶

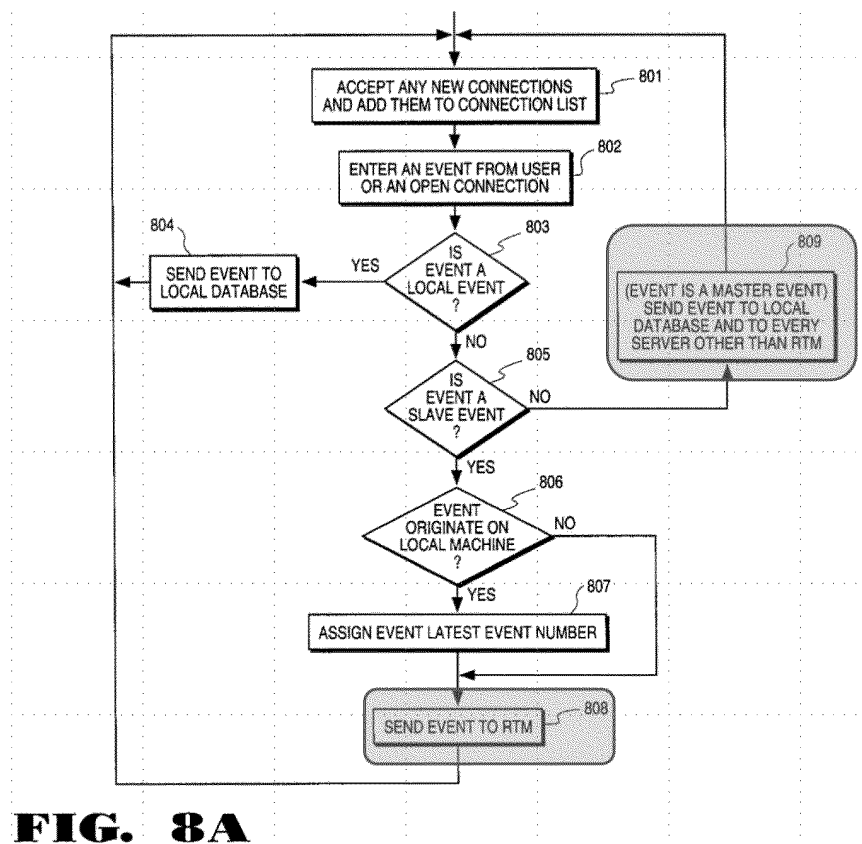


FIG. 8A

⁶⁴⁵ Roy; 2:38-41.

⁶⁴⁶ Roy; 1:66-2:10 (highlighting added).

“ FIG. 8A illustrates a routine used by a slave event server for opening connections to other event servers and for routing events. Initially, the slave event server accepts any new connections and adds them to its connection list (step 801). Next, the slave event server inputs an event received either from a user or from one of its open connections (step 802). If (in step 803) the event is a local event (i.e., if the event is intended only for use on the local machine), then the slave event server sends the event only to its local database (step 804) and the routine repeats from the beginning. If the event is not a local event and the event is a slave event (step 805) which originated on the local machine (step 806), then the event is assigned the latest event number for the local machine (step 807) and is routed to the RTM (step 808). . . . If the event is neither a local event nor a slave event, then the event is a master event, in which case the event is routed to the local database and to every server connection other than the RTM (step 809).”⁶⁴⁷

[1A] configuring the server to store a model associated with the application and to execute view-generating and controller logic associated with the application; and

503. In my opinion, Roy discloses configuring the server (e.g., central server 61 and/or the computer with the master event server and master database) to store a model associated with the application (e.g., a database representing the world model for the game) and to execute view-generating and controller logic associated with the application, as those terms have been construed. Thus, in my opinion (and as explained below), Roy discloses:

- a. configuring the server (e.g., central server 61 or the computer with the master event server and master database) to store the data, rules, and algorithms affecting the data associated with the application (e.g., a database representing the world model for the game);
- b. configuring the server (e.g., central server 61 or the computer with the master event server and master database) to execute program code (e.g., program code known to a person with ordinary skill in the art at the time the ‘ 719 Patent was filed) that processes user requests (e.g., a user enters a command and generates an “ event”) and causes the model to be changed and/or the view to be refreshed (e.g., the event causes the database representing the world model to be changed and the user’ s screen is ultimately refreshed to reflect the result of their command); and
- c. configuring the server (e.g., central server 61 and/or the computer with the master event server and master database) to execute program code (program code known to a person with ordinary skill in the art at the time the ‘ 719 Patent was filed) that generates a screen or window representation of a subset of the model that the application chooses to

⁶⁴⁷ Roy; Figure 8A (annotated, highlighting added) and 6:46-67 (highlighting added).

display (e.g., the server generates an updated view of the game world as affected by the user's command).

504. First, Roy discloses configuring the server to store a model associated with the application. For example, Roy configures the “master database” on the server to store a “databas represent[ing] a world model for the game.”⁶⁴⁸ Roy further explains how the “game world can be modeled by a database” which is located on a “central server.”⁶⁴⁹

“A method and apparatus for reducing network latency during execution of a multiple-player game across a computer network are provided. A master database represents a world model for the game, and a master event server sequences user inputs, or events, for updating the master database. The master event server and the master database are initially located on a particular node in the network. Every other node that is used by a player in the game has a slave event server and a slave database.”⁶⁵⁰

505. Second, Roy discloses configuring the server to execute view-generating logic associated with the application. As an initial matter, Roy discusses the concept of “[r]endering commands” ~~that~~ [r] sent from the updated database 53 in the central server 51 across the network to update the displays 55 and 57 of client computers 50 and 52, respectively.”⁶⁵¹ Roy discloses that the central server 61 sends rendering commands to one or more client computers (e.g., computers 60 and 62).⁶⁵² While the *transmission* of rendering commands is primarily discussed in relation to the prior art embodiment illustrated in Figures 4A and 4B, Roy acknowledges that “[b]y providing slave copies of the database in every other machine involved in the game, the need to send rendering commands across the network is *reduced* or eliminated, which reduces bandwidth consumption.”⁶⁵³ Thus, while the invention disclosed in Roy may *reduce*

⁶⁴⁸ Roy; Abstract, 4:8-40.

⁶⁴⁹ Roy; 1:28-30.

⁶⁵⁰ Roy; Abstract (highlighting added).

⁶⁵¹ Roy; Figure 4B and 4:23-29 (emphasis added).

⁶⁵² Roy; Figure 4B and 4:23-29.

⁶⁵³ Roy; 4:50-54.

transmitting rendering commands across the network, that necessarily means some rendering commands are still transmitted (and therefore the server still executes view-generating logic).

506. For this discussion, I will focus on rendering commands that are not transmitted (i.e., where the view-generating logic that produces the rendering commands is executed on the same computer where the view is to be displayed). The most complete description of this functionality is provided in relation to Figure 5A, where, focusing on computer 62, Roy explains that “[t]he display 74 of computer system 62 is updated by rendering commands 73 from the slave copy 72 of master database 71.”⁶⁵⁴ These rendering commands (views) are generated by program code in slave copy 72. As Roy explains:

“FIG. 5A illustrates a technique according to the present invention for implementing a multi-user game over the Internet. In FIG. 5A, the game is played by two players, players A and B, operating computer systems 60 and 62, respectively. It is noted, however, that the present invention is equally applicable to a game played simultaneously by more than two players. Initially, the master database 71 and the master event server 66 are located in a central server computer system 61. Located in computer system 60 (used by player A) are a slave event queue 65 and a slave copy 68 of master database 71. Similarly, located in computer system 62 (used by player B) is a slave event server 67 and a slave copy 72 of master database 71. The display 74 of computer system 62 is updated by rendering commands 73 from the slave copy 72 of master database 71, rather than from the master database 71 itself, thereby reducing bandwidth consumption on the network. Similarly, the display 70 associated with computer system 60 is updated by rendering commands 69 from slave copy 68 of master database 71, rather than directly from master database 71. Events 64 from computer system 62 are received by slave event server 67 and then passed along to the master event server 66, unless the events 64 are purely local events for updating only the slave copy 72 of the database. Purely local events might be, for example, events which set display options or other functions that are only relevant to computer system 62. Similarly, events 63 originating from computer system 60 are received first by slave event server 65 and then passed along to master event server 66, unless those events are purely local to computer system 60. Events received by the master event server 66 are used to update the master database 71 and are passed along to all slave event servers as master events 75.”⁶⁵⁵

507. Roy also discloses that whenever a computer becomes an effective server (i.e., the master event server and master database are migrated to that computer, thus resulting in that

⁶⁵⁴ Roy, 5:29-32.

⁶⁵⁵ Roy, 5:17-49.

computer being the “server” of the claim), then that computer executes view-generating logic to generate the views for the player of that computer.⁶⁵⁶ For example, in Figure 6, computer 62 has become an effective **server** and contains the master event server and the master database.⁶⁵⁷ It receives events from player B on computer 62 (and from other players), updates master database 72, and generates a view (as reflected in rendering commands 73) to display on screen 74 of computer 62.⁶⁵⁸ This functionality is described in relation to Figure 5A above, and is illustrated in Figure 6 here:

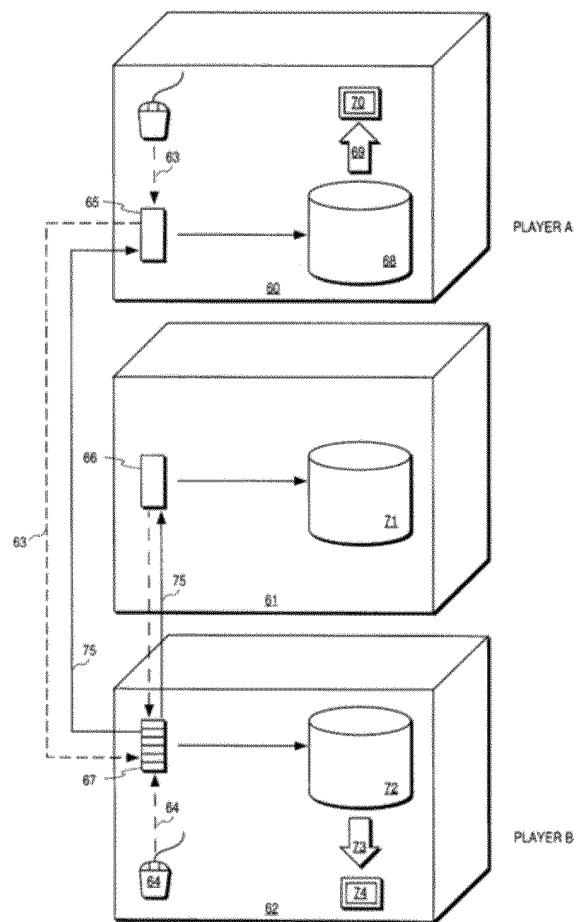


FIG. 6

⁶⁵⁶ Roy; Figure 6 and 4:50-55, 7:1-8.

⁶⁵⁷ Roy; 6:1-11.

⁶⁵⁸ Roy; Figure 6 and 4:50-55, 5:27-36, 7:1-8.

508. To summarize, when the master event server and master database have been migrated to computer 62, computer 62 acts as a server in this client/server system. And computer 62 executes view-generating logic when it creates the rendering commands used to update the display 74.

509. Third, Roy discloses the server executing controller logic associated with the application. For example, Roy explains that the “ master event server sequences user inputs, or events, for updating the master database” (i.e., controller logic is executed to update the game world model):

“ A method and apparatus for reducing network latency during execution of a multiple-player game across a computer network are provided. A master database represents a world model for the game, and a master event server sequences user inputs, or events, for updating the master database. The master event server and the master database are initially located on a particular node in the network. Every other node that is used by a player in the game has a slave event server and a slave database.”⁶⁵⁹

510. As background, Roy discloses how the user can issue commands and thereby generate “ events” using a mouse, joystick, trackball, keyboard, or any other I/O device. as depicted by Figure 4B, software 54 on the central server 51 “ receives events 58 from computers 50 and 52 and provides these events to the database 53 in order to update the game world” and, as described, send rendering commands back to client computers 50 and 52 to reflect the result of the user’s command (i.e., the server executes controller logic to update the game world model stored in database 53 and causes the view to be refreshed on client computers 50 and 52).⁶⁶¹

511. Continuing with the example I discuss above in relation to Figure 6, computer 62 has become an effective server and receives events from player B on computer 62. Computer 62 then executes controller logic to update master database 72 and cause rendering commands 73 to

⁶⁵⁹ Roy; Abstract (highlighting added).

⁶⁶⁰ Roy; 3:66-4:13.

⁶⁶¹ Roy; 4:23-29.

be rendered and shown to the user on his display 74.⁶⁶² Similarly, player B may issue commands that would be processed entirely locally by effective server computer 62. In this case, computer 62 executes controller logic to update master database 72 and cause rendering commands 73 to be rendered and shown to the user on his display 74. In both of these cases, because computer 62 is acting as the server in the client/server system (since the master event server and master database are migrated to that computer), Roy discloses configuring the server to store and execute controller logic.

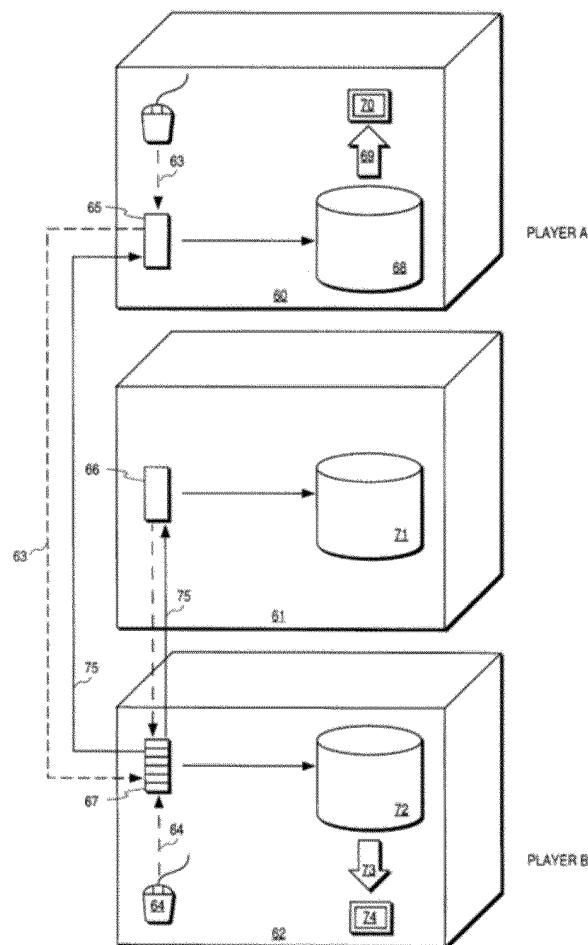


FIG. 6

⁶⁶² Roy; Figure 6 and 5:27-49.

*Obvious over Roy in light of Bennett*⁶⁶³

512. To the extent Roy does not expressly disclose this limitation, it is further my opinion that it would have been obvious to a person of ordinary skill in the art at the time the Patent was filed to modify the online multiplayer gaming system described in Roy based on Bennett's teachings about building distributed applications using objects. 719

513. A POSITA would have been motivated to modify the interactive online applications described in Roy based on Bennett's teachings. For example, since Smalltalk-80 represented the canonical implementation of the Model-View-Controller (MVC) paradigm, a POSITA would have been motivated to utilize a distributed implementation of Smalltalk such as Bennett when implementing any distributed system using the MVC paradigm.

514. Further, Roy and Bennett are in the same field of endeavor (e.g., distributed applications).

515. In addition, a paper entitled "The Design and Implementation of Distributed Smalltalk"⁶⁶⁴ was published in the Proceedings of the 1987 ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA). A POSITA working in the field of distributed object-based systems who was a member of the Association for Computing Machinery (ACM) – the largest organization of computer scientists and engineers in the world – would have likely received by mail a printed copy of the OOPSLA conference proceedings in which this paper was published. Joining the ACM was a typical step in becoming a professional computer scientist or engineer, and joining SIGPLAN (the Special Interest Group on Programming

⁶⁶³ J.K. Bennett, *Experience with Distributed Smalltalk*, Software—Practice and Experience, Vol. 20(2), pp. 157-180, Feb. 1990 ("Bennett").

⁶⁶⁴ J.K. Bennett, *The Design and Implementation of Distributed Smalltalk*, Proceedings of the Second ACM Conference on Object-Oriented Programming Systems, Languages, and Applications, Orlando, Florida, October 1987, Special Issue of SIGPLAN Notices, 22, (12), pp. 318-330 (1987).

[REDACTED]

Languages) would have been a common step for those aspiring to work in distributed object-oriented systems. All members of SIGPLAN received a printed copy of the OOPSLA conference proceedings as a membership benefit.

516. A POSITA examining Roy and the online multiplayer gaming system described therein would be motivated to modify the client/server architecture of a game depending on the specific requirements for that game.

517. A POSITA examining Roy and the online gaming system described therein would be motivated to modify the client/server architecture of the game depending on the specific requirements for that game. For example, a POSITA would understand that certain types or genres of games would require a different architecture design depending on their processing requirements. As such, a POSITA would seek to distribute the program data and logic effectively and efficiently between the server and client.

518. Bennett discloses a method that enables an application designer to distribute objects containing data and logic (including view-generating logic and controller logic) between servers and client devices, and a POSITA would be motivated to use Bennett's methods.

519. For example, Distributed Smalltalk " provides communication and interaction among geographically remote Smalltalk users, direct access to remote objects, the ability to construct distributed applications in the Smalltalk environment, and a degree of object sharing among users."⁶⁶⁵ " Potential applications of Distributed Smalltalk include mail systems, remote computation servers, remote file servers, and collaborative software development."⁶⁶⁶ " The

⁶⁶⁵ Bennett; p. 157.

⁶⁶⁶ Bennett; p. 157.

distributed aspects of the system are largely user transparent and preserve the reactive quality of Smalltalk objects.”⁶⁶⁷

520. As another example, as described by the ‘ 719 Patent, applications can be designed “ in terms of the MVC (Model-View-Controller) paradigm” by using the programming language Smalltalk.⁶⁶⁸ “ Smalltalk is an object-oriented programming language and highly interactive uniformly object-structured programming environment.”⁶⁶⁹

521. The ‘ 719 Patent incorporates by reference “ A cookbook for using the model-view controller user interface paradigm in Smalltalk-80”⁶⁷⁰ by Glenn E. Krasner and Stephen T. Pope.⁶⁷¹ This paper was subsequently revised slightly and published as “ A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System” (“ Pope”). In terms of content, the two papers are nearly identical. “ Pope,” as used in this report, refers to the revised (and easier to read) version of the paper. Pope teaches about “ the Model-View-Controller (MVC) programming paradigm and methodology used in the Smalltalk-80 programming system.”⁶⁷² Bennett used Smalltalk-80 as its implementation language. “ The [object-oriented programming] of these classes are part of the Smalltalk-80 . . . a registered trademark of ParcPlace Systems.”⁶⁷³ Pope describes the benefits of using Smalltalk: “ The user interface of the Smalltalk-80 programming environment . . . was developed using a particular strategy of representing information, display, and control.”⁶⁷⁴ “ This strategy was chosen to satisfy two goals: (1) to create the special set of system components needed to support a highly interactive software development process, and (2) to provide a general

⁶⁶⁷ Bennett; p. 157.

⁶⁶⁸ ‘ 719 Patent, 1:34-35, Background.

⁶⁶⁹ Bennett; p. 157.

⁶⁷⁰ G. Krasner and S. Pope, *A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80*; Journal of Object-Oriented Programming, SIGS Publications, Vol. 1, No. 3, pp. 26-49, Aug. 1988.

⁶⁷¹ ‘ 719 Patent, 1:35-39.

⁶⁷² Pope; Abstract, p. 1.

⁶⁷³ Bennett; p. 161.

⁶⁷⁴ Pope; p. 2.

set of system components that make it possible for programmers to create portable interactive graphical applications easily.”⁶⁷⁵ Pope further describes the benefits of MVC programming. For example, see the excerpt from Pope shown below:⁶⁷⁶

MVC and the Issues of Reusability and Pluggability

When building interactive applications, as with other programs, modularity of components has enormous benefits. Isolating functional units from each other as much as possible makes it easier for the application designer to understand and modify each particular unit, without having to know everything about the other units. Our experiences with the Smalltalk-76 programming system showed that one particular form of modularity--a three-way separation of application components--has payoff beyond merely making the designer's life easier. This three-way division of an application entails separating (1) the parts that represent the model of the underlying application domain from (2) the way the model is presented to the user and from (3) the way the user interacts with it.

Model-View-Controller (MVC) programming is the application of this three-way factoring, whereby objects of different classes take over the operations related to the application domain (the model), the display of the application's state (the view), and the user interaction with the model and the view (the controller). In earlier Smalltalk system user interfaces, the tools that were put into the interface tended to consist of arrangements of four basic viewing idioms: paragraphs of text, lists of text (menus), choice "buttons," and graphical forms (bit- or pixel-maps). These tools also tended to use three basic user interaction paradigms: browsing, inspecting and editing. A goal of the current Smalltalk-80 system was to be able to define user interface components for handling these idioms and paradigms once, and share them among all the programming environment tools and user-written applications using the methodology of MVC programming.

522. Bennett's Distributed Smalltalk “ is an implementation of ~~Smalltalk~~⁶⁷⁷,” because a POSITA would understand that Smalltalk could be used to design applications using an MVC paradigm, a POSITA would similarly understand that Distributed Smalltalk could be used to design distributed applications using an MVC paradigm.

⁶⁷⁵ Pope; p. 2

⁶⁷⁶ Pope; p. 2.

⁶⁷⁷ Bennett; p. 157.

[REDACTED]

523. At least in light of the foregoing, a POSITA would be motivated to create and/or modify the online interactive applications described by Roy by using Distributed Smalltalk. For example, a POSITA may seek to “ offload[] a compute-intensive task” to the server.⁶⁷⁸ Similarly, a POSITA could distribute DS objects containing data and logic (including model data, view-generation logic, or controller logic) between the client and server depending on the requirements for a specific application, and would understand that the application could be implemented per the MVC paradigm in view of Bennett’s use of Smalltalk. This is particularly true since Roy was equally concerned with distributing processing between the computer systems.⁶⁷⁹

[1B] configuring the client to store at least a subset of the model associated with the application and to execute at least a subset of the view-generating and controller logic associated with the application;

524. In my opinion, Roy discloses configuring the client (e.g., computer system 60) to store at least a subset of the model associated with the application (e.g., a subset of the database comprising the world model for the online multiplayer application) and to execute at least a subset of the view-generating and controller logic associated with the application, as those terms have been construed. Thus, in my opinion (and as explained below), Roy discloses:

- a. configuring the client (e.g., computer system 60) to store at least a subset of the data, rules, and algorithms affecting the data associated with the application (e.g., at least a subset of the world model stored in the master database is migrated to each client computer);
- b. configuring the client (e.g., computer system 60) to execute program code (program code known to a POSITA at the time the ’ 719 Patent was filed) that processes user requests (e.g., a user command that generates an “ event”) and causes the model to be changed and/or the view to be refreshed (e.g., the event causes the local database containing a subset of the game world data to be changed and the view is refreshed on computer system 60 to reflect the change);
- c. configuring the client (e.g., computer system 60) to execute program code (program code known to a POSITA at the time the ’ 719 Patent was filed) that generates a screen

⁶⁷⁸ Bennett; p. 157.

⁶⁷⁹ Roy; 4:18-54.

or window representation of a subset of the model that the application chooses to display (e.g., client computer system 60 generates an updated view that is displayed to the user of computer system 60); and

- d. that the view-generating logic and controller logic executed on the client is the same view-generating logic and controller logic executed on the server, or some subset thereof (e.g., the *same* view-generating logic and controller logic would be executed on each computer system connected to the game network).

525. First, Roy discloses configuring the client to store at least a subset of the model associated with the application. In particular, Roy discloses that “[s]lave copies of the master database (or portions thereof) are stored”⁶⁸⁰ on each client computer. As I explained previously, the databases of Roy represent the world model for the game. So because the clients contain a copy of the master database (or a portion thereof), it necessarily means that the client is configured to store at least a subset of the model. Further, as Roy explains:

“ A method and apparatus for reducing network latency during execution of a multiple-player game across a computer network are provided. A master database represents a world model for the game, and a master event server sequences user inputs, or events, for updating the master database. The master event server and the master database are initially located on a particular node in the network. Every other node that is used by a player in the game has a slave event server and a slave database.”⁶⁸¹

⁶⁸⁰ Roy; 4:44-48.

⁶⁸¹ Roy; Abstract (highlighting added).

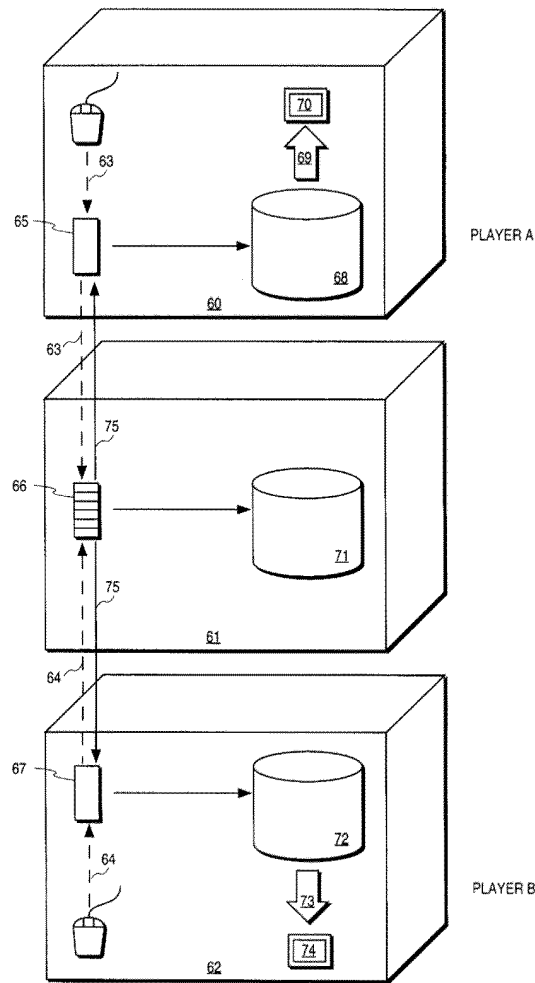


FIG. 5A

“ FIG. 5A illustrates a technique according to the present invention for implementing a multi-user game over the Internet. In FIG. 5A, the game is played by two players, players A and B, operating computer systems 60 and 62, respectively. It is noted, however, that the present invention is equally applicable to a game played simultaneously by more than two players. Initially, the master database 71 and the master event server 66 are located in a central server computer system 61. Located in computer system 60 (used by player A) are a slave event queue 65 and a slave copy 68 of master database 71. Similarly, located in computer system 62 (used by player B) is a slave event server 67 and a slave copy 72 of master database 71. The display 74 of computer system 62 is updated by rendering commands 73 from the slave copy 72 of master database 71, rather than from the master database 71 itself, thereby reducing bandwidth consumption on the network. Similarly, the display 70 associated with computer system 60 is updated by rendering commands 69 from slave copy 68 of master database 71, rather than directly from master database 71. Events 64 from computer system 62 are received by slave event server 67 and then passed along to the master event server 66, unless the events 64 are purely local events for updating only the slave copy 72 of the database. Purely local events might be, for example, events which set display options or other functions that are only relevant to computer system 62. Similarly, events 63 originating from computer system 60 are

received first by slave event server 65 and then passed along to master event server 66, unless those events are purely local to computer system 60. Events received by the master event server 66 are used to update the master database 71 and are passed along to all slave event servers as master events 75.”⁶⁸²

526. Thus, as an example, the entire master database “ or portions thereof⁶⁸³⁻⁶⁷ would be saved in database 68 of client computer 60, meaning at least a subset of the data comprising the world model would be stored on the client. (This is true even when the master database and master event server have been migrated to computer 62, which is the scenario I discuss in claim element 1A.)

527. Second, Roy discloses configuring the client to execute at least a subset of the view-generating logic associated with the application. As an example, Roy discloses that the display 70 of client computer 60 is updated by rendering commands 69 from the slave copy 68 of master database 72 (when the master database has been migrated to computer 62), rather than from the master database itself, thereby reducing bandwidth consumption on the network:

⁶⁸² Roy; Figure 5A and 5:17-49 (highlighting added).

⁶⁸³ Roy; 4:47-48.

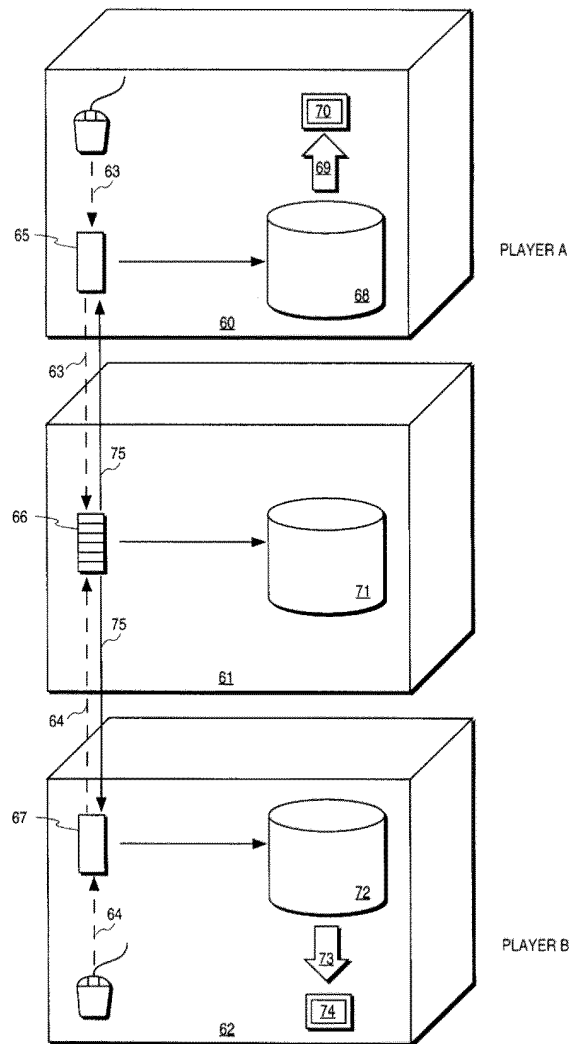


FIG. 5A

“ FIG. 5A illustrates a technique according to the present invention for implementing a multi-user game over the Internet. In FIG. 5A, the game is played by two players, players A and B, operating computer systems 60 and 62, respectively. It is noted, however, that the present invention is equally applicable to a game played simultaneously by more than two players. Initially, the master database 71 and the master event server 66 are located in a central server computer System 61. Located in computer system 60 (used by player A) are a slave event queue 65 and a slave copy 68 of master database 71. Similarly, located in computer system 62 (used by player B) is a slave event server 67 and a slave copy 72 of master database 71. The display 74 of computer system 62 is updated by rendering commands 73 from the slave copy 72 of master database 71, rather than from the master database 71 itself, thereby reducing bandwidth consumption on the network. Similarly, the display 70 associated with computer system 60 is updated by rendering commands 69 from slave copy 68 of master database 71, rather than directly from master database 71. Events 64 from computer system 62 are received by slave event server 67 and then passed along to the master event server 66, unless the events 64 are purely local events for updating only the slave copy 72 of the database. Purely local

events might be, for example, events which set display options or other functions that are only relevant to computer system 62. Similarly, events 63 originating from computer system 60 are received first by slave event server 65 and then passed along to master event server 66, unless those events are purely local to computer system 60. Events received by the master event server 66 are used to update the master database 71 and are passed along to all slave event servers as master events 75.”⁶⁸⁴

528. As a specific example, Roy discloses client computer system 60 executing at least a subset of the view-generating logic. As described above, slave copies of the database (or a portion thereof) are migrated to every client computer because this reduces “ the need to send rendering commands across the network.”⁶⁸⁵ Thus, each client computer may execute view-generating logic to create a view for each respective player because each player has at least a subset of the world model stored on their respective computers.⁶⁸⁶ Further, it would be understood to a POSITA, or at least rendered obvious, that the view-generating logic would be the same on each computer for efficiency and economic reasons. It would be particularly obvious to a POSITA given that Roy specifically addresses online multiplayer games wherein each player sees the *same* world. It would not make sense to use different logic to generate views of the same world for each computer.

529. As just one simple example, imagine two players connected to an online multiplayer game as described by Roy. Each controls an avatar that can explore the virtual world. If, for example, the two players were standing in the exact same location and were looking at the virtual world from the same perspective, then each player’s screen would display the same view. In this case, it would make little sense if the view-generating logic on each player’s computer was different. A POSITA would not endeavor to do redundant work and coding for the same functionality.

⁶⁸⁴ Roy; Figure 5A and 5:17-49 (highlighting added).

⁶⁸⁵ Roy; 4:51-54.

⁶⁸⁶ Roy; 5:29-33 (“ The display 74 of computer system 62 is updated by rendering commands 73 from the slave copy 72 of master database 71, rather than from the master database 71 itself.”).

[REDACTED]

530. That the program code on each computer would be the same (or at least that it would be obvious) is also evident by the very nature of commercial multi-player games. The program code for the game running on player A's computer is the same as the program code for the game running player B's computer. This is particularly true where, as Roy explains, any node may become an effective server—each player's computer would need the functionality to run the game (and serve as the “master”). As such a person of ordinary skill in the art would understand (or at the very least find it obvious) that program code used to operate the game (e.g., generate the rendering commands for display) would be the same across the player's computers.

531. Third, Roy discloses configuring the client to execute at least a subset of the controller logic associated with the application on the client. For example, Roy discloses that the client executes controller logic to process local events and cause the model of the world stored in the local database to be updated and the view to be refreshed for that user. Roy explains:

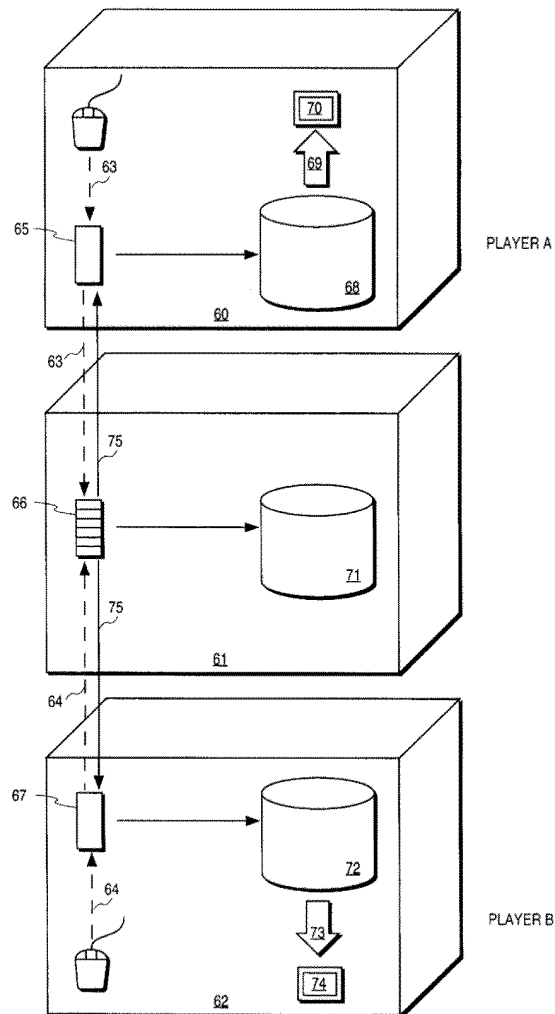


FIG. 5A

“ FIG. 5A illustrates a technique according to the present invention for implementing a multi-user game over the Internet. In FIG. 5A, the game is played by two players, players A and B, operating computer Systems 60 and 62, respectively. It is noted, however, that the present invention is equally applicable to a game played simultaneously by more than two players. Initially, the master database 71 and the master event server 66 are located in a central server computer system 61. Located in computer system 60 (used by player A) are a slave event queue 65 and a slave copy 68 of master database 71. Similarly, located in computer system 62 (used by player B) is a slave event server 67 and a slave copy 72 of master database 71. The display 74 of computer system 62 is updated by rendering commands 73 from the slave copy 72 of master database 71, rather than from the master database 71 itself, thereby reducing bandwidth consumption on the network. Similarly, the display 70 associated with computer system 60 is updated by rendering commands 69 from slave copy 68 of master database 71, rather than directly from master database 71. Events 64 from computer system 62 are received by slave event server 67 and then passed along to the master event server 66, unless the events 64 are purely local events for updating only the slave copy 72 of the database. Purely local events might be, for example, events which set display options or other functions that are only

relevant to computer system 62. Similarly, events 63 originating from computer system 60 are received first by slave event server 65 and then passed along to master event server 66, unless those events are purely local to computer system 60. Events received by the master event server 66 are used to update the master database 71 and are passed along to all slave event servers as master events 75.”⁶⁸⁷

532. As a specific example, Roy discloses that client computer system 60 executes at least a subset of the controller logic. As described, client computer 60 processes events generated by user commands and determines whether the events should be passed along to the server, or whether they are “local events” that are used to update the local model stored in database 68 and cause display 70 to be refreshed.⁶⁸⁸ This determination is done by an event server 65 on computer system 60. Every computer system connected to the game has an “event server” and each is tasked with “routing events to the local database and to other event servers.”⁶⁸⁹ And the master event server can be moved between the central server 61 and any computer system.⁶⁹⁰ Thus, the *same* controller logic can be executed on any client computer.

533. Additionally, for the same reasons described above regarding the view-generating logic, it would be understood by a POSITA, or would at least be obvious to a POSITA, that the controller logic on each computer system would be the same. For example, it would be obvious to a POSITA that each computer would process “purely local events” in the same way, such as local events for setting display options.⁶⁹¹ Further, because any computer can become an effective server, a POSITA would understand that the controller logic on each computer would be the same, or at the very least it would be obvious to do so for the reasons discussed above.

⁶⁸⁷ Roy; Figure 5A and 5:17-49 (highlighting added).

⁶⁸⁸ Roy; 5:17-49.

⁶⁸⁹ Roy; 4:62-66.

⁶⁹⁰ Roy; 6:1-11.

⁶⁹¹ Roy; 5:40-43.

***Obvious over Roy in light of Rover*⁶⁹²**

534. To the extent Roy does not expressly disclose this limitation, it is further my opinion that it would have been obvious to a person of ordinary skill in the art at the time the ' 719 Patent was filed to modify the online multiplayer gaming system described in Roy based on Rover' s teachings about moving application data and logic between a client and server using relocatable dynamic objects (" RDOs").

535. A POSITA would have been motivated to modify the online gaming system described in Roy based on Rover' s teachings. For example, Roy and Rover are in the same field of endeavor (interactive distributed systems).

536. In addition, Rover was published in the leading venue for operating and distributed systems research (the annual Symposium on Operating Systems Principles (SOSP). A POSITA working in the field of distributed systems who was a member of the Association for Computing Machinery (ACM) – the largest organization of computer scientists and engineers in the world – would have likely received by post mail a printed copy of the SOSP conference proceedings in which Rover was published.

537. Joining the ACM was a typical step in becoming a professional computer scientist or engineer, and joining SIGOPS (the Special Interest Group on Operating Systems) would have been a common step for those aspiring to work in distributed systems. All members of SIGOPS received a printed copy of the SOSP conference proceedings as a membership benefit.

538. A POSITA examining Roy and the online gaming system described therein would be motivated to modify the client/server architecture of the game depending on the specific requirements for that game. For example, a POSITA would understand that certain types or genres

⁶⁹² A. Joseph et al., *Rover: A Toolkit for Mobile Information Access*, Proc. 15th ACM SOSP, pp. 156-171, Dec. 1995.

[REDACTED]

of games would require a different architecture design depending on their processing requirements. As such, a POSITA would seek to distribute the program data and logic effectively and efficiently between the server and client. Rover discloses a client/server architecture that utilizes RDOs to move logic between the server and client as desired by the application designer.

539. For example, Rover explains that “[t]he Rover toolkit combines relocatable dynamic objects and queued remote procedure calls to provide unique services for ‘roving’ mobile applications.”⁶⁹³ “A relocatable dynamic object is an object with a well-defined interface that can be dynamically loaded into a client computer from a server computer (or vice versa) to reduce client-server communication requirements.”⁶⁹⁴ “The Rover toolkit also offers advantages for workstation applications by providing a uniform distributed object architecture for code shipping, object caching, and asynchronous object invocation.”⁶⁹⁵ “Depending on the power of the mobile host and the available bandwidth, Rover dynamically adapts and moves functionality between the client and the server.”⁶⁹⁶ “By migrating a GUI to the client, input events can be processed locally, even if the client becomes disconnected.”⁶⁹⁷

540. In light of at least the foregoing, a POSITA would understand that data and application logic (including view-generating and controller logic) for the online gaming system described in Roy could be allocated to either the server or the client, and would be motivated to incorporate Rover’s method for distributing that data and logic (including model data, controller logic, and view-generating logic) because, “[d]epending on the power of the [client]

⁶⁹³ Rover; p. 156.

⁶⁹⁴ Rover; p. 156.

⁶⁹⁵ Rover; p. 156.

⁶⁹⁶ Rover; p. 158.

⁶⁹⁷ Rover; p. 159.

[REDACTED]

and the available bandwidth, Rover dynamically adapts and moves functionality between the client and the server.”⁶⁹⁸

Obvious over Roy in light of Bennett⁶⁹⁹

541. To the extent Roy does not expressly disclose this limitation, it is further my opinion that it would have been obvious to a person of ordinary skill in the art at the time the’ 719 Patent was filed to modify the online multiplayer gaming system described in Roy based on Bennett’ s teachings about building distributed applications using objects.

542. A POSITA would have been motivated to modify the interactive online applications described in Roy based on Bennett’ s teachings. For example, since Smalltalk-80 represented the canonical implementation of the Model-View-Controller (MVC) paradigm, a POSITA would have been motivated to utilize a distributed implementation of Smalltalk such as Bennett when implementing any distributed system using the MVC paradigm.

543. Further, Roy and Bennett are in the same field of endeavor (e.g., distributed applications).

544. In addition, a paper entitled “ The Design and Implementation of Distributed Smalltalk”⁷⁰⁰ was published in the Proceedings of the 1987 ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA). A POSITA working in the field of distributed object-based systems who was a member of the Association for Computing Machinery (ACM) – the largest organization of computer scientists and engineers in the world – would have likely received by mail a printed copy of the OOPSLA conference proceedings in

⁶⁹⁸ Rover, p. 158.

⁶⁹⁹ J.K. Bennett, *Experience with Distributed Smalltalk*, Software— Practice and Experience, Vol. 20(2), pp. 157-180, Feb. 1990 (“ Bennett”).

⁷⁰⁰ J.K. Bennett, *The Design and Implementation of Distributed Smalltalk*, Proceedings of the Second ACM Conference on Object-Oriented Programming Systems, Languages, and Applications, Orlando, Florida, October 1987, Special Issue of SIGPLAN Notices, 22, (12), pp. 318-330 (1987).

[REDACTED]

which this paper was published. Joining the ACM was a typical step in becoming a professional computer scientist or engineer, and joining SIGPLAN (the Special Interest Group on Programming Languages) would have been a common step for those aspiring to work in distributed object-oriented systems. All members of SIGPLAN received a printed copy of the OOPSLA conference proceedings as a membership benefit.

545. A POSITA examining Roy and the online multiplayer gaming system described therein would be motivated to modify the client/server architecture of a game depending on the specific requirements for that game.

546. A POSITA examining Roy and the online gaming system described therein would be motivated to modify the client/server architecture of the game depending on the specific requirements for that game. For example, a POSITA would understand that certain types or genres of games would require a different architecture design depending on their processing requirements. As such, a POSITA would seek to distribute the program data and logic effectively and efficiently between the server and client.

547. Bennett discloses a method that enables an application designer to distribute objects containing data and logic (including view-generating logic and controller logic) between servers and client devices, and a POSITA would be motivated to use Bennett's methods.

548. For example, Distributed Smalltalk " provides communication and interaction among geographically remote Smalltalk users, direct access to remote objects, the ability to construct distributed applications in the Smalltalk environment, and a degree of object sharing among users."⁷⁰¹ " Potential applications of Distributed Smalltalk include mail systems, remote computation servers, remote file servers, and collaborative software development."⁷⁰² " The

⁷⁰¹ Bennett; p. 157.

⁷⁰² Bennett; p. 157.

distributed aspects of the system are largely user transparent and preserve the reactive quality of Smalltalk objects.”⁷⁰³

549. As another example, as described by the ‘ 719 Patent, applications can be designed “ in terms of the MVC (Model-View-Controller) paradigm” by using the programming language Smalltalk.⁷⁰⁴ “ Smalltalk is an object-oriented programming language and highly interactive uniformly object-structured programming environment.”⁷⁰⁵

550. The ‘ 719 Patent incorporates by reference “ A cookbook for using the model-view controller user interface paradigm in Smalltalk-80”⁷⁰⁶ by Glenn E. Krasner and Stephen T. Pope.⁷⁰⁷ This paper was subsequently revised slightly and published as “ A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System” (“ Pope”). In terms of content, the two papers are nearly identical. “ Pope,” as used in this report, refers to the revised (and easier to read) version of the paper. Pope teaches about “ the Model-View-Controller (MVC) programming paradigm and methodology used in the Smalltalk-80 programming system.”⁷⁰⁸ Bennett used Smalltalk-80 as its implementation language. “ The [object-oriented programming] of these classes are part of the Smalltalk-80 . . . a registered trademark of ParcPlace Systems.”⁷⁰⁹ Pope describes the benefits of using Smalltalk: “ The user interface of the Smalltalk-80 programming environment . . . was developed using a particular strategy of representing information, display, and control.”⁷¹⁰ “ This strategy was chosen to satisfy two goals: (1) to create the special set of system components needed to support a highly interactive software development process, and (2) to provide a general

⁷⁰³ Bennett; p. 157.

⁷⁰⁴ ‘ 719 Patent, 1:34-35, Background.

⁷⁰⁵ Bennett; p. 157.

⁷⁰⁶ G. Krasner and S. Pope, *A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80*; Journal of Object-Oriented Programming, SIGS Publications, Vol. 1, No. 3, pp. 26-49, Aug. 1988.

⁷⁰⁷ ‘ 719 Patent, 1:35-39.

⁷⁰⁸ Pope; Abstract, p. 1.

⁷⁰⁹ Bennett; p. 161.

⁷¹⁰ Pope; p. 2.

set of system components that make it possible for programmers to create portable interactive graphical applications easily.”⁷¹¹ Pope further describes the benefits of MVC programming. For example, see the excerpt from Pope shown below:⁷¹²

MVC and the Issues of Reusability and Pluggability

When building interactive applications, as with other programs, modularity of components has enormous benefits. Isolating functional units from each other as much as possible makes it easier for the application designer to understand and modify each particular unit, without having to know everything about the other units. Our experiences with the Smalltalk-76 programming system showed that one particular form of modularity--a three-way separation of application components--has payoff beyond merely making the designer's life easier. This three-way division of an application entails separating (1) the parts that represent the model of the underlying application domain from (2) the way the model is presented to the user and from (3) the way the user interacts with it.

Model-View-Controller (MVC) programming is the application of this three-way factoring, whereby objects of different classes take over the operations related to the application domain (the model), the display of the application's state (the view), and the user interaction with the model and the view (the controller). In earlier Smalltalk system user interfaces, the tools that were put into the interface tended to consist of arrangements of four basic viewing idioms: paragraphs of text, lists of text (menus), choice "buttons," and graphical forms (bit- or pixel-maps). These tools also tended to use three basic user interaction paradigms: browsing, inspecting and editing. A goal of the current Smalltalk-80 system was to be able to define user interface components for handling these idioms and paradigms once, and share them among all the programming environment tools and user-written applications using the methodology of MVC programming.

551. Bennett's Distributed Smalltalk “ is an implementation of Smalltalk.”⁷¹³ because a POSITA would understand that Smalltalk could be used to design applications using an MVC paradigm, a POSITA would similarly understand that Distributed Smalltalk could be used to design distributed applications using an MVC paradigm.

⁷¹¹ Pope; p. 2

⁷¹² Pope; p. 2.

⁷¹³ Bennett; p. 157.

552. At least in light of the foregoing, a POSITA would be motivated to create and/or modify the online interactive applications described by Roy by using Distributed Smalltalk. For example, a POSITA may seek to “ offload[] a compute-intensive task” to the server.⁷¹⁴ Similarly, a POSITA could distribute DS objects containing data and logic (including model data, view-generation logic, or controller logic) between the client and server depending on the requirements for a specific application, and would understand that the application could be implemented per the MVC paradigm in view of Bennett’s use of Smalltalk.

[1C] wherein one or more portions of the application are performed at the client without the client having to interact with the server, and

553. In my opinion, as described above regarding claim preamble 1P, Roy discloses wherein one or more portions of the application (e.g., the online multiplayer game) are performed at the client (e.g., computer system 60) without the client having to interact with the server (e.g., the computer system that has the master event server and master database). I incorporate those explanations here.

554. Roy further discloses:

⁷¹⁴ Bennett; p. 157.

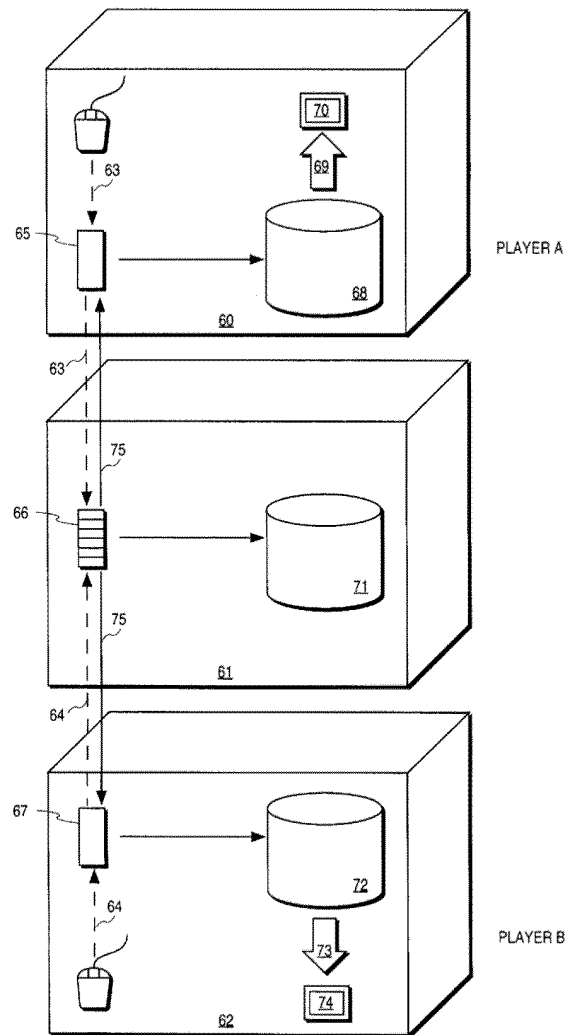
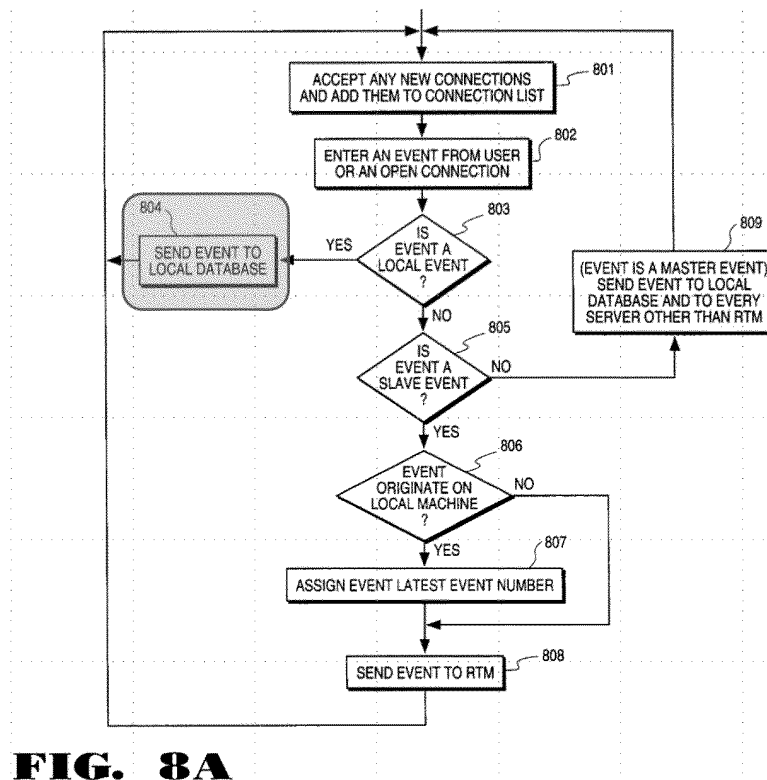


FIG. 5A

“ FIG. 5A illustrates a technique according to the present invention for implementing a multi-user game over the Internet. In FIG. 5A, the game is played by two players, players A and B, operating computer systems 60 and 62, respectively. It is noted, however, that the present invention is equally applicable to a game played simultaneously by more than two players. Initially, the master database 71 and the master event server 66 are located in a central server computer system 61. Located in computer system 60 (used by player A) are a slave event queue 65 and a slave copy 68 of master database 71. Similarly, located in computer system 62 (used by player B) is a slave event server 67 and a slave copy 72 of master database 71. The display 74 of computer system 62 is updated by rendering commands 73 from the slave copy 72 of master database 71, rather than from the master database 71 itself, thereby reducing bandwidth consumption on the network. Similarly, the display 70 associated with computer system 60 is updated by rendering commands 69 from slave copy 68 of master database 71, rather than directly from master database 71. Events 64 from computer system 62 are received by slave event server 67 and then passed along to the master event server 66, unless the events 64 are purely local events for updating only the slave copy 72 of the database. Purely local

events might be, for example, events which set display options or other functions that are only relevant to computer system 62. Similarly, events 63 originating from computer system 60 are received first by slave event server 65 and then passed along to master event server 66, unless those events are purely local to computer system 60. Events received by the master event server 66 are used to update the master database 71 and are passed along to all slave event servers as master events 75.”⁷¹⁵



“ FIG. 8A illustrates a routine used by a slave event server for opening connections to other event servers and for routing events. Initially, the slave event server accepts any new connections and adds them to its connection list (step 801). Next, the slave event server inputs an event received either from a user or from one of its open connections (step 802). If (in step 803) the event is a local event (i.e., if the event is intended only for use on the local machine), then the slave event server sends the event only to its local database (step 804) and the routine repeats from the beginning. If the event is not a local event and the event is a slave event (step 805) which originated on the local machine (step 806), then the event is assigned the latest event number for the local machine (step 807) and is routed to the RTM (step 808). . . . If the event is neither a local event nor a slave event, then the event is a master event, in which case the event is routed to the local database and to every server connection other than the RTM (step 809)” ⁷¹⁶

⁷¹⁵ Roy; Figure 5A and 5:17-49 (highlighting added).

⁷¹⁶ Roy; Figure 8A (annotated, highlighting added) and 6:46-67 (highlighting added).

555. As an example, and as described above, “ local events” are processed at the client without having to interact with the server.⁷¹⁷

[1D] further wherein the client and the server both locally maintain at least a portion of the model and execute the view-generating and controller logic associated therewith.

556. In my opinion, Roy discloses that the client and the server both locally maintain at least a portion of the model (e.g., a portion of the game world model), and execute the view-generating and controller logic associated therewith, as described above regarding claim elements 1A and 1B. I incorporate those explanations here.

557. I explained in my descriptions of the model, view-generating logic, and controller logic for claim elements 1A and 1B how: (1) both the client and server maintain at least a portion of the model (e.g., a portion of the game world model), (2) the client and server execute the same controller logic (e.g., the client and server execute the same logic to process the same event created by a user command to update their respective local world models and/or to cause their respective screens to be refreshed) and (3) the client and server execute the same view-generating logic (e.g., the client and server computers execute the same view-generating logic when showing a view of the same world to the respective players on each computer). I incorporate those explanations here.

c. Roy Renders Obvious Claim 2 of the ’ 719 Patent

[2] The method of claim 1, wherein the client and server communicate over a HyperText Transport Protocol [HTTP] network.

558. As discussed above, Roy anticipates or renders obvious claim 1 of the ’ 719 Patent.

559. In my opinion, Roy renders obvious wherein the client and server communicate over an HTTP network. For example, as detailed above, Roy discloses a method for solving latency

⁷¹⁷ Roy; Figure 8A and 5:29-46.

issues for multiplayer games that are played *on the Internet*.⁷¹⁸ It would be obvious to a POSITA at the time the ' 719 Patent was filed that HTTP would (or could) be used for the client/server system described by Roy to communicate over the Internet.

560. Indeed, it would be obvious to a POSITA that *any* client/server system from the relevant time could utilize HTTP, as discussed in Section 8.B.7.a.

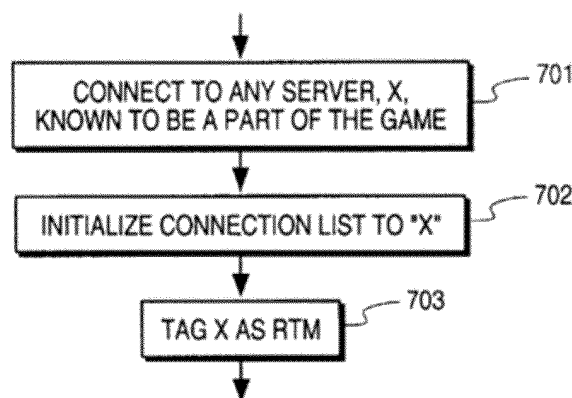
d. Roy Anticipates or Renders Obvious Claim 12 of the ' 719 Patent

[12] The method of claim 1, wherein the at least a subset of the model, the view-generating and the controller logic associated with the application are downloaded from the server to the client upon demand.

561. As discussed above, Roy anticipates or renders obvious claim 1 of the ' 719 Patent.

562. In my opinion, Roy discloses or renders obvious wherein the at least a subset of the model, the view-generating and the controller logic associated with the application are downloaded from the server to the client upon demand.

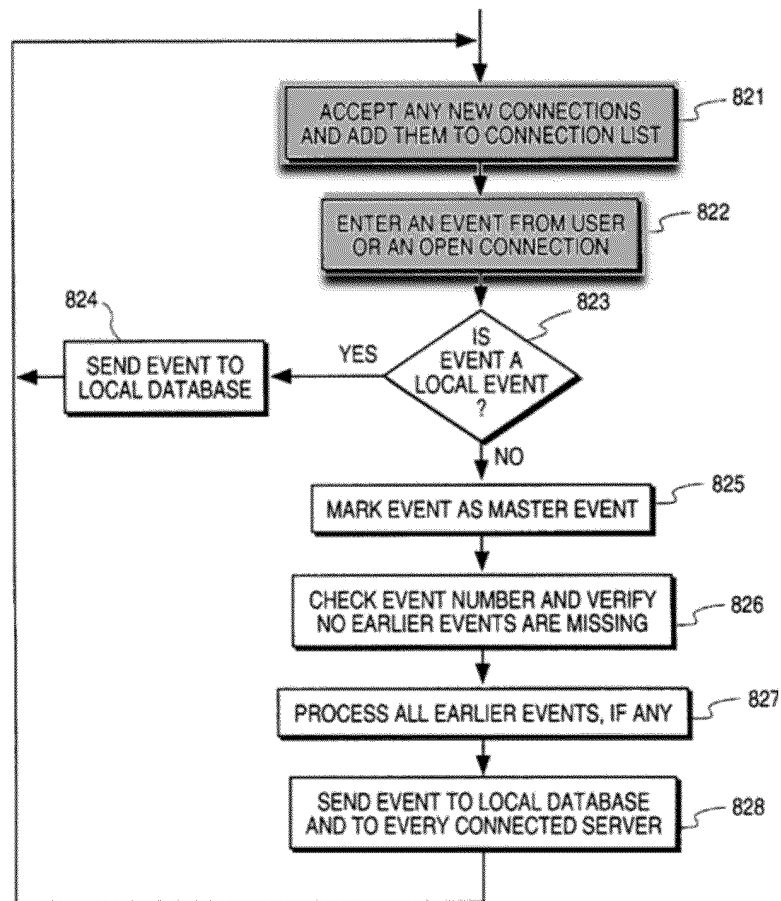
563. To start, Roy discloses how client computers may connect to a server to join the online game. For example, Roy discloses “ a technique for establishing a connection to a computer system having multi-player game software.”⁷¹⁹ This is depicted by Figure 7:



⁷¹⁸ Roy; 1:19-22 (“ [T]here are unique problems associated with implementing a multi-player computer game across a [WAN], particularly one as expansive as the Internet.”).

⁷¹⁹ Roy; Figure 7 and 2:41-43.

564. Similarly, Figure 8B “ illustrates a routine used by a master event server for opening connections.”⁷²⁰



565. As another example, Figure 6 depicts how client computers 60 and 62 access an online application at central server 61. Or, as Roy describes, “ FIG. 6 illustrates a multi-player game running in a network in which a slave event server has established a direct connection to the master event server.”⁷²¹

566. Additionally, Roy discloses that server data (including the model) can be migrated to any client computer on demand:

⁷²⁰ Roy; Figure 8B (highlighting added) and 7:1-2.

⁷²¹ Roy; 2:38-41.

“ Regardless of which device has the actual authority to approve of migration of the master event server, the migration process can be initiated by a request or command transmitted over the network by a slave event server. The request or command is a special event known as a ‘ slave migration event.’ The master event server initiates the migration process by sending a ‘ master migration event’ . When a slave event server receives a slave migration event from another slave event server, it adds its identifier to the end of the event message. Hence, the slave migration event gets annotated with the complete path it follows to reach the master event server. The master migration event sent in reply also contains the same path information. Accordingly, interceding slave event servers can correctly route the master migration event to the slave event server that transmitted the slave migration event and update their respective RTM information.”⁷²²

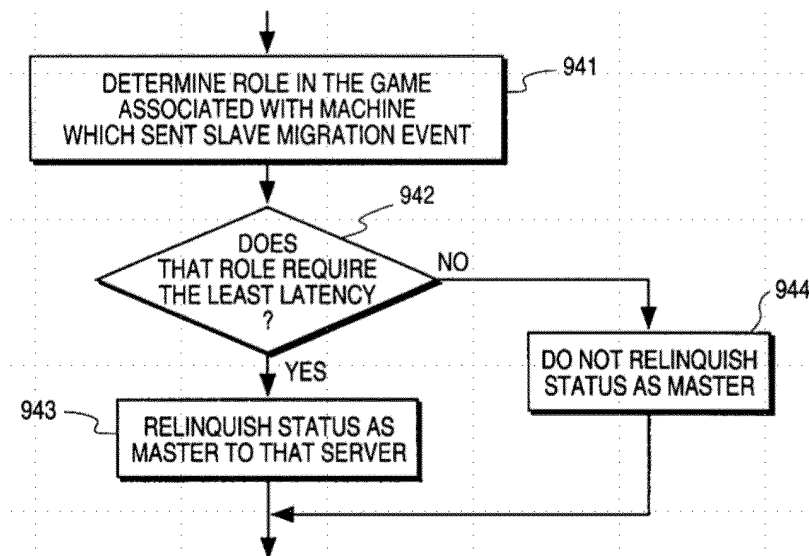


FIG. 9C

“ FIG. 9C illustrates a routine by which a master event server can determine whether to relinquish its status as master event server. Upon receiving a slave migration event, the master event server identifies the role in the game that is associated with the slave event server that sent the slave migration event (step 941). If that role requires the lowest latency of any role in the game, then it is appropriate to migrate the master event server. Accordingly, the master event server relinquishes its status as master event server (step 943). Otherwise, the master event server retains its status as master (step 944).”⁷²³

567. In light of the foregoing, it would be understood by or at least rendered obvious to a POSITA that, when a new user joined the online multiplayer game, a slave copy of the database would be transferred to the new user’s computer (i.e., at least a subset of the model would be

⁷²² Roy; 7:43-59 (highlighting added).

⁷²³ Roy; Figure 9C and 8:41-51 (highlighting added).

[REDACTED]

downloaded from the server to the client upon demand). Indeed, *every* client computer obtained a slave copy of the model at some point.⁷²⁴ Roy also explains that the event server of a new player can “connect[] to another event server that is already part of the game.”⁷²⁵

568. It would similarly be obvious to a POSITA that— at this same time that a subset of the model is downloaded— a subset of the view-generating logic and controller logic could also be downloaded.

569. Clients downloading software from servers has been practiced for as long as it has been possible to do so. Since at least the mid 1990s, this was the most common method of software upgrade. Operating systems such as Microsoft Windows 95 (released in 1995) routinely used the ability of user machines (clients) to download updates upon request from Microsoft servers to keep those systems up-to-date.

570. Another example of clients downloading view-generating and controller logic associated with an application from the server would be Java applets, introduced in 1995 by Sun Microsystems. Java applets were small applications typically written in the Java programming language, and then compiled into what were called Java “bytecodes.” These bytecodes were then downloaded by a browser, and executed by an instance of the Java Virtual Machine (a separate program that ran in concert with the web browser). The user could launch a Java applet from a web page, or a Java applet could appear in a frame of the web page.

571. Applets were used to provide interactive features to web applications that could not be provided by HTML alone. They could capture keyboard and mouse input, which could then be used to update the displayed graphic content.

⁷²⁴ Roy; 4:51-55.

⁷²⁵ Roy; 5:50-64.

[REDACTED]

572. JavaScript, introduced in late 1995 by Netscape Navigator, and Jscript, introduced by Microsoft in 1996, are yet other examples of clients downloading view-generating and controller logic associated with an application from the server. As described in Section 6.G, JavaScript and Jscript allowed web developers to create web pages with client-side interactivity. Used in conjunction with HTML (HyperText Markup Language) and CSS (Cascading Style Sheets), JavaScript and Jscript provided the means to create dynamic webpage behavior (this combination was often referred to as Dynamic HTML, or DHTML).

573. In the 1990s, website developers routinely use JavaScript or Jscript to create code for client-side executable content. Web browsers of the time had a built-in JavaScript (or Jscript) engine that executed the downloaded code within the browser on the user's machine. This code, which typically included both view-generation and controller logic, was downloaded with its associated web page upon demand (when the website is visited within the browser).

574. These examples demonstrate that a POSITA in the relevant time frame would have a thorough understanding of the practice of downloading view-generating and controller logic from the server to the client upon demand. A POSITA wishing to implement a web-based application in the late 1990s would have routinely used this understanding in such an implementation.

575. Thus, a POSITA examining Roy would understand that the controller and view-generating logic could be downloaded to a client computer connecting to the game network. For example, a POSITA would understand that downloading all of the data and logic together would be more efficient than requiring multiple client/server interactions. Plus, it would be obvious to a POSITA to include the logic needed to process the data along with the data being sent. And in any case, the earliest point that data or logic could be transmitted would be after the client connected.

[REDACTED]

The only alternative would be if the player already had some of the data and/or logic on a hard disk (or previously installed from a hard disk). Indeed, a POSITA would understand that, at the relevant time, the only other option to deliver the data and logic to a user's computer would be via CD-ROM. Given what are essentially two options (sending the data and logic online or delivering it physically with a disk), a POSITA would be motivated to at least consider both options.

Obvious over Roy in light of Rover

576. To the extent Roy does not expressly disclose this limitation, it is further my opinion that it would have been obvious to a person of ordinary skill in the art at the time the ' 719 Patent was filed to modify the online multiplayer gaming system described in Roy based on Rover's teachings about moving application data and logic between a client and server on demand using relocatable dynamic objects (" RDOs").

577. A POSITA would have been motivated to modify the online multiplayer gaming system described by Roy based on Rover's teachings. For example, Roy and Rover are in the same field of endeavor (interactive distributed systems).

578. In particular, a POSITA examining Roy and the online multiplayer game system described therein would need to determine how to distribute game data and logic to each client computer. At the relevant time, a POSITA would understand that the logic could either be sent on a disk of some sort (e.g., CD-ROM) or sent over a network. A POSITA would be motivated to consider both options.

579. Rover teaches a method for downloading data and application logic over the network on demand. For example, Rover explains that "[t]he Rover toolkit combines relocatable dynamic objects and queued remote procedure calls to provide unique services for ' roving' mobile

applications.”⁷²⁶ “ A relocatable dynamic object is an object with a well-defined interface that can be dynamically loaded into a client computer from a server computer (or vice versa) to reduce client-server communication requirements.”⁷²⁷ “ The Rover toolkit also offers advantages for workstation applications by providing a uniform distributed object architecture for code shipping, object caching, and asynchronous object invocation.”⁷²⁸ “ Depending on the power of the mobile host and the available bandwidth, Rover dynamically adapts and moves functionality between the client and the server.”⁷²⁹ For example, “ [b]y migrating a GUI to the client, input events can be processed locally, even if the client becomes disconnected.”⁷³⁰ As another example, Rover describes an e-mail system “ which is contained in RDOs retrieved from an HTTP server when the user first loads the browser” — i.e., on⁷³¹ demand.

580. Rover further explains how to handle a situation wherein the client connection to a server is lost, which a POSITA would be motivated to incorporate into the online gaming system described by Roy to handle situations wherein one of the clients are disconnected. For example: “ Queued remote procedure call is a communication system that permits applications to continue to make non-blocking remote procedure call requests even when a host is disconnected, with requests and responses being exchanged upon network reconnection.”⁷³²

581. A POSITA would be motivated to incorporate Rover’s methods for downloading a subset of logic over a network into the online gaming system described by Roy to download data and program logic from the server to each client device rather than, for example, require use of a

⁷²⁶ Rover; p. 156.

⁷²⁷ Rover; p. 156.

⁷²⁸ Rover; p. 156.

⁷²⁹ Rover; p. 158.

⁷³⁰ Rover; p. 159.

⁷³¹ Rover; p. 165.

⁷³² Rover; p. 156.

CD-ROM. A POSITA would also be motivated to utilize Rover's teachings concerning how to handle a client that becomes disconnected, as described above.

e. Roy Anticipates or Renders Obvious Claim 26 of the ' 719 Patent

[26P] A method for use in a client/server system of reducing interactions between a client and server in association with an application being accessed by the client at the server, the method comprising the steps of:

582. To the extent the preamble is considered to be limiting, in my opinion, Roy discloses or renders obvious a method for use in a client/server system of reducing interactions between a client and server in association with an application being accessed by the client at the server, for the reasons described above for claim preamble 1P and claim element 1C. I incorporate those discussions by reference here.

[26A] configuring the server to:

- (i) store a model associated with the application;**
- (ii) execute view-generating logic associated with the application; and**
- (iii) execute controller logic associated with the application; and**

583. In my opinion, Roy discloses configuring the server (e.g., computer 62 – when it has the master event server and master database) to store a model associated with the application (e.g., a database representing the world model for the game) and to execute view-generating and controller logic associated with the application, for the reasons described above for claim element 1A. I incorporate that discussion by reference here.

[26B] configuring the client to:

- (i) store at least a subset of the model associated with the application;**
- (ii) execute at least a subset of the view-generating logic associated with the application; and**
- (iii) execute at least a subset of the controller logic associated with the application;**

584. In my opinion, Roy discloses configuring the client (e.g., computer system 60) to store at least a subset of the model associated with the application (e.g., a subset of the database

[REDACTED]

comprising the world model for the online multiplayer application) and to execute at least a subset of the view-generating and controller logic associated with the application, for the reasons described above for claim 1B. I incorporate that discussion by reference here.

[26C] wherein one or more portions of the application are performed at the client without the client having to interact with the server,

585. In my opinion, Roy that one or more portions of the application are performed at the client without the client having to interact with the server, for the reasons explained above for limitation 1C. I incorporate that discussion by reference here.

[26D] and further wherein the client and the server both locally maintain at least a portion of the model and execute the view-generating and controller logic associated therewith;

586. In my opinion, Roy discloses that the client and the server both locally maintain at least a portion of the model and execute the view-generating and controller logic associated therewith, for the reasons explained above for limitations 1A, 1B, and 1D. I incorporate those discussions by reference here.

[26E] further wherein, in accordance with such a dual model-view-controller arrangement, a model comprises application data, rules, and algorithms affecting the data, a view comprises a screen or window representation of a subset of the model that the application chooses to display, and a controller comprises the logic that processes user requests, and causes the model to be changed and the view to be refreshed.

587. In my opinion, Roy discloses that, in accordance with a dual model-view-controller arrangement, a model comprises application data, rules, and algorithms affecting the data, a view comprises a screen or window representation of a subset of the model that the application chooses to display, and a controller comprises the logic that processes user requests, and causes the model to be changed and the view to be refreshed, for the reasons explained above for limitations 1A, 1B, and 1D. I incorporate those discussions by reference here.

[REDACTED]

588. I incorporate the discussions of limitations 1A, 1B, and 1D in their entirety here, but to summarize: Roy discloses a model that comprises application data, rules, and algorithms affecting the data (e.g., a database representing the world model for the game); Roy discloses a view that comprises a screen or window representation of a subset of the model that the application chooses to display (e.g., the view displayed on the monitors of the current virtual world); Roy discloses a controller comprising the logic that processes user requests, and causes the model to be changed and the view to be refreshed (e.g., logic that processes a user command and generates an “ event” , which is then displayed).

7. Additional Opinions Relating to Dependent Claims 2 and 12

a. Asserted Claim 2 of the ’ 719 Patent Would be Obvious to a POSITA

589. In my opinion, the additional limitation recited in Asserted Claim 2 of the ’ 719 Patent (“ The method of claim 1, wherein the client and server communicate over a HyperText Transport Protocol network.”) would be obvious to a person of ordinary skill in the art at the time the ’ 719 Patent was filed. Indeed, a POSITA from the relevant time would understand that a client/server system would (or could) utilize HTTP for communication. The bases of this opinion are set forth below.

590. According to Internet Live Stats, there were 17,087,182 websites in the year 2000.⁷³³ Virtually all of these websites were accessed using HTTP, since HTTP was the default Internet protocol used for this purpose. It is not an exaggeration to say that in the year 2000, accessing a website was synonymous with using HTTP. Any POSITA would have expected to use HTTP when developing a web-based application. No POSITA would have reasonably considered using HTTP in a web-based application as novel or innovative.

⁷³³ <https://www.internetlivestats.com/total-number-of-websites/>.

[REDACTED]

591. A POSITA would also have known and understood that the HTTP protocol was simply an application-layer protocol that employed the transport layer TCP protocol. A POSITA would have also known and understood the relationship between HTTP, TCP, and UDP, as described in Section 6.F. In 2000, TCP was routinely used for reliable in-order packet delivery on large area networks like the World Wide Web; UDP was used on small local area networks where performance goals outweighed reliability concerns. Making these kinds of choices was (and still is) routine practice in the design of distributed systems.

592. As I have already described, it is further my opinion that, to the extent that any of the prior art cited in this report is found not to disclose using HTTP in their client/server systems, a POSITA would have found it trivial to do so.

593. For example, Distributed Smalltalk employed UDP datagrams on the small local area network that connected machines in the Department of Computer Science at the University of Washington in 1987 (see Section 8.B.4). Reliable delivery of packets was not a concern in this case, since network traffic was light, and the number of machines limited. Had reliability been a concern, changing one line of code would have caused Distributed Smalltalk to use TCP instead of UDP (see Section 8.A.4).

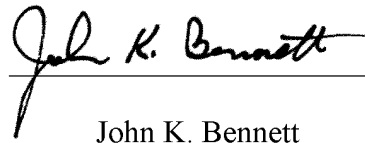
594. A POSITA would have routinely used HTTP to implement web-based applications in the late 1990s, and could have created web-based applications based on the teachings of any prior art reference cited in this report (e.g., the online multiplayer games described in Roy and Yelon, the online bookstore application described in Lowery, the Rover Applications described in Rover, the distributed applications described in Bennett, etc.). The fact that the networking components of these systems used TCP or UDP as the transport layer would have made the networking aspects of such an undertaking trivial, as the example in Section 8.A.4 demonstrates.

[REDACTED]

639. I understand that expert discovery is ongoing. I reserve the right to respond to any reports that are submitted by Plaintiff's expert witnesses or to any testimony by Plaintiff's fact or expert witnesses, whether at deposition or at trial. My opinions are subject to change based on additional information provided or positions taken by Plaintiff or its experts, or based on additional work I may perform. If appropriate, I may amend or supplement my opinions in a follow-on report.

640. I declare under penalty of perjury that the foregoing is true and correct to the best of my knowledge. If called upon to testify, I would do so consistent with the statements and opinions contained in this report, and in any follow-on reports.

Executed on February 2, 2024


John K. Bennett