

Filed: January 31, 2025

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

CARVANA, LLC
Petitioner,

v.

INTERNATIONAL BUSINESS MACHINES CORPORATION
Patent Owner.

Inter Partes Review No.: IPR2025-00564

U.S. Patent No. 7,702,719

**PETITION FOR *INTER PARTES* REVIEW
OF U.S. PATENT NO. 7,702,719**

TABLE OF CONTENTS

	<u>Page</u>
I. MANDATORY NOTICES	1
A. Real Party in Interest	1
B. Related Matters.....	1
C. Lead and Back-up Counsel	2
D. Service Information	2
II. GROUNDS AND REQUESTED RELIEF	2
III. STANDING	3
IV. TECHNOLOGY	3
A. The Aged Technology of MVC and “Dual-MVC”	3
B. JavaScript Was Known at Least as of 1995	6
C. The ’719 Patent Claims the Well-Known “Dual-MVC” Paradigm.....	7
1. Overview	7
2. Level of Ordinary Skill in the Art.....	11
3. Claim Construction	12
D. Prior Art.....	14
1. U.S. Patent No. 5,838,909 (Roy)	14
a. Roy Is Analogous Prior Art.....	14
b. Roy Discloses a “Dual-MVC” Networked System.....	15
2. U.S. Patent No. 5,695,400 (Fennell).....	20
a. Fennell Is Analogous Prior Art.....	20
b. Fennell Relates to Roy and Discloses Distributed Internet Applications Accessed Via Browser Software.....	21
3. <i>JavaScript: The Definitive Guide</i> (O’Reilly)	23
a. O’Reilly Is a Printed Publication and Analogous Prior Art.....	23
b. O’Reilly Demonstrates Familiarity with HTTP as of at Least 1997	24

c.	O'Reilly Discloses Using Frames and Invisible Frames to Generate HTML Documents and Associate View-Generating and Controller Logic, the Same Subject Matter Purported to Be (But Was Not) Invented in the '719 Patent	25
d.	O'Reilly Discloses the Use of APIs	27
e.	O'Reilly Discloses Using JavaScript to Create and Modify HTML Documents in Browsers, Including Interaction with Java—the Same Subject Matter Claimed in the '719 Patent	27
f.	O'Reilly Describes JavaScript's Caching Functions, the Same Functions Claimed as "Application-Independent" in the '719 Patent	28
g.	O'Reilly Discloses "Event Handlers" that Provide Controller Logic.....	29
V.	THE '719 PATENT IS INVALID.....	30
A.	Ground 1: Roy Anticipates Claims 1, 13, 25, and 26	30
1.	Preamble.....	30
2.	Limitation (a)	33
3.	Limitation (b)	36
4.	Limitation (c)	41
5.	Limitation (d)	43
6.	Limitation (e)	44
B.	Ground 2: Roy Renders Obvious Claims 1, 13, 25, and 26.....	44
C.	Ground 3: Roy, Fennell, and O'Reilly Render Obvious Dependent Claims 2, 4–12, 14, 16–24	46
1.	Motivation to Combine	46
a.	Roy and Fennell.....	46
b.	Roy and O'Reilly.....	50
c.	Roy, Fennell, and O'Reilly.....	52
2.	Claims 2 and 14: "a HyperText Transport Protocol"	53
3.	Claims 4 and 16: "partition/ing a screen area associated with the browser software into frames"	54
4.	Claims 5 and 17: "wherein the at least a subset of the model, the view-generating and the controller logic associated with the application are associated with at least one frame and one or more views for display in	

	accordance with the application are associated with at least another frame”	56
5.	Claims 6 and 18: “wherein the at least one view frame is a visible frame”	60
6.	Claims 7 and 19: “wherein the at least one frame associated with the at least a subset of the model, the view-generating logic and the controller logic is not a visible frame”	61
7.	Claims 8 and 20: “form/ing at least one frame with which application-independent view-generating logic and controller logic is associated”	62
8.	Claims 9 and 21: “wherein the at least one application-independent view-generating logic and controller logic frame further has an application-independent model associated therewith”	67
9.	Claims 10 and 22: “wherein the at least one application-independent view-generating logic and controller logic frame serves as an application programming interface for developing views to be displayed in accordance with the application”	70
10.	Claims 11 and 23: “wherein the views are implemented in accordance with the HyperText Markup Language and the application programming interface is implemented in accordance with the JavaScript language”	73
11.	Claims 12 and 24: “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”	74
D.	Ground 4: Roy and Fennell Render Obvious Dependent Claims 3 and 15 Requiring “Browser Software”	76
VI.	SECONDARY CONSIDERATIONS	77
VII.	DISCRETIONARY DENIAL UNDER <i>FINTIV</i> IS INAPPROPRIATE	80
VIII.	DISCRETIONARY DENIAL UNDER §325(d) IS INAPPROPRIATE	80
IX.	CONCLUSION.....	80

TABLE OF AUTHORITIES

	Page(s)
Cases	
<i>Apple Inc. v. Fintiv, Inc.</i> , IPR2020-00019 (PTAB Mar. 20, 2020)	80
<i>Apple, Inc. v. Gesture Tech. Partners, LLC</i> , IPR2021-00922, Paper 10 (PTAB Nov. 29, 2021).....	7
<i>B/E Aerospace, Inc. v. C&D Zodiac, Inc.</i> , 962 F.3d 1373 (Fed. Cir. 2020)	45
<i>Ethicon LLC v. Intuitive Surgical, Inc.</i> , No. 2021-1601, 2022 WL 1576779 (Fed. Cir. 2022)	14
<i>Geotab USA, Inc. v. Omega Patents, LLC</i> , 2024 Pat. App. LEXIS 2628, IPR2023-00504, Paper 45 (P.T.A.B. July 18, 2024).....	79
<i>Hulu, LLC v. Sound View Innovations, LLC</i> , IPR2018-01039, Paper 29 (PTAB Dec. 20, 2019)	23
<i>Merck & Cie v. Gnosis S.p.A.</i> , 808 F.3d 829 (Fed. Cir. 2015)	77, 79
<i>Realtime Data, LLC v. Iancu</i> , 912 F.3d 1368 (Fed. Cir. 2019)	44
<i>Robert Bosch, LLC v. Iancu</i> , 778 Fed. App'x 871 (Fed. Cir. 2019)	77, 79

TABLE OF EXHIBITS

Exhibit No.	Description
1001	U.S. Patent No. 7,702,719 (“the ’719 Patent”)
1002	File history of U.S. Patent No. 7,702,719 (Application No. 09/500,208)
1003	Declaration of John K. Bennett, Ph.D., P.E. (“Bennett Decl.”)
1004	U.S. Patent No. 5,838,909 (“Roy”)
1005	U.S. Patent No. 5,695,400 (“Fennell”)
1006	JAVASCRIPT: THE DEFINITIVE GUIDE (Flanagan, David) (Willison, Frank, Ed.) (O’Reilly & Associates, Inc., pub.) (2d. Ed. Jan. 1997) (ISBN: 1-56592-234-4) (“O’Reilly”)
1007	U.S. Patent No. 5,768,510 (“Gish”)
1008	G. Krasner & S. Pope, <i>A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System</i> , Journal of Object-Oriented Programming, Vol. 1, Jan. 1988 (“Krasner”)
1009	John K. Bennett, The Design and Implementation of Distributed Smalltalk, In <i>Proceedings of the Second ACM Conference on Object-Oriented Programming Systems, Languages, and Applications</i> , Orlando, Florida, October 1987. Special Issue of SIGPLAN Notices, Volume 22, Number 12, December, 1987 (“Bennett”)
1010	D. Decouchant, Design of a Distributed Object Manager for the Smalltalk-80 System. In <i>Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications</i> , Portland, Oregon, October 1986. Special Issue of SIGPLAN Notices, Volume 21, Number 11, November, 1986 (“Decouchant”)

1011	P. L. McCullough, Transparent Forwarding: First Steps, In <i>Proceedings of the Second ACM Conference on Object-Oriented Programming Systems, Languages, and Applications</i> , Orlando, Florida, October 1987. Special Issue of SIGPLAN Notices, Volume 22, Number 12, December, 1987 (“McCullough”)
1012	Maurice Wilkes, Slave Memories and Dynamic Storage Allocation, IEEE Transactions on Electronic Computers, Vol. EC-1.4, Issue: 2, (April 1965) 270-271 (“Wilkes”)
1013	U.S. Patent No. 3,771,137 (“Barner”)
1014	Betz, et al., <i>Developing Highly-Responsive User Interfaces with DHTML and Servlets</i> , RC 21628 (Dec. 20, 1998)
1015	<i>Sotera</i> Stipulation
1016	<i>International Business Machines Corp. v. Zynga Inc.</i> , Case No. 22-cv-590-GBW, D.I. 529 (D. Del.)
1017	<i>International Business Machines Corp. v. Zynga Inc.</i> , Case No. 22-cv-590-GBW, Excerpts of Sept. 12, 2024 Trial Tr. (D. Del.)

CLAIMS

Independent Claims

	Claim 1	Claim 13	Claim 25	Claim 26
Preamble	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	A network-based system, comprising	An article of manufacture for use in a client/server system for reducing interactions between a client and server in association with an application being accessed by the client at the server, comprising machine readable media containing one or more programs which when executed implement the steps of:	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
(a)	configuring the server to store a model associated with the application and to execute view-generating and	a server having at least one processor operative to: (i) store a model associated with an application	configuring the server to store a model associated with the application and to execute view-generating and controller logic	configuring the server to: (i) store a model associated with the application; (ii) execute view-generating logic associated with the application;

	Claim 1	Claim 13	Claim 25	Claim 26
	controller logic associated with the application; and	associated with the server, and (ii) execute view-generating and controller logic associated with the application; and	associated with the application; and	and (iii) execute controller logic associated with the application; and
(b)	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;	a client, coupled to the server via a network, having at least one processor operative to: (i) store at least a subset of the model associated with the application; and (ii) execute at least a subset of the view-generating and controller logic associated with the application;	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;	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;
(c)	wherein one or more portions	wherein one or more portions	wherein one or more portions	wherein one or more portions of

	Claim 1	Claim 13	Claim 25	Claim 26
	of the application are performed at the client without the client having to interact with the server, and	of the application are performed at the client without the client having to interact with the server such that interactions between the client and server are reduced, and	of the application are performed at the client without the client having to interact with the server, and	the application are performed at the client without the client having to interact with the server, and
(d)	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.	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.	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.	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;
(e)				further wherein, in accordance with such a dual model-view-controller arrangement, a model comprises

	Claim 1	Claim 13	Claim 25	Claim 26
				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.

Dependent Claims

Claim 2	Claim 14
The method of claim 1, wherein the client and server communicate over a HyperText Transport Protocol network.	The system of claim 13, wherein the network is a HyperText Transport Protocol network.
Claim 3	Claim 15
The method of claim 1, wherein the client performs the one or more portions of the application in accordance with browser software running thereon.	The system of claim 13, wherein the client processor performs the one or more portions of the application in accordance with browser software running thereon.
Claim 4	Claim 16
The method of claim 3, wherein the configuring step further comprises the step of partitioning a screen area associated with the browser software into frames.	The system of claim 15, wherein the client processor is further operative to partition a screen area associated with the browser software into frames.
Claim 5	Claim 17
The method of claim 4, wherein the at least a subset of the model, the view-generating and the controller logic associated with the application are associated with at least one frame and one or more views for display in accordance with the application are associated with at least another frame.	The system of claim 16, wherein the at least a subset of the model, the view-generating and the controller logic associated with the application are associated with at least one frame and one or more views for display in accordance with the application are associated with at least another frame.
Claim 6	Claim 18

The method of claim 5, wherein the at least one view frame is a visible frame.	The system of claim 17, wherein the at least one view frame is a visible frame.
Claim 7	Claim 19
The method of claim 5, wherein the at least one frame associated with the at least a subset of the model, the view-generating logic and the controller logic is not a visible frame.	The system of claim 17, wherein the at least one frame associated with the at least a subset of the model, the view generating logic and the controller logic is not a visible frame.
Claim 8	Claim 20
The method of claim 4, wherein the configuring step further comprises forming at least one frame with which application-independent view-generating logic and controller logic is associated.	The system of claim 16, wherein the client processor is further operative to format least one frame with which application-independent view-generating logic and controller logic is associated.
Claim 9	Claim 21
The method of claim 8, wherein the at least one application-independent view-generating logic and controller logic frame further has an application-independent model associated therewith.	The system of claim 20, wherein the at least one application-independent view-generating logic and controller logic frame further has an application-independent model associated therewith.
Claim 10	Claim 22
The method of claim 8, wherein the at least one application-independent view-generating logic and controller logic frame serves as an application programming interface for developing views to be displayed in accordance with the application.	The system of claim 20, wherein the at least one application-independent view-generating logic and controller logic frame serves as an application programming interface for developing views to be displayed in accordance with the application.

Claim 11	Claim 23
The method of claim 10, wherein the views are implemented in accordance with the HyperTextMarkup Language and the application programming interface is implemented in accordance with the JavaScript language.	The system of claim 22, wherein the views are implemented in accordance with the HyperTextMarkup Language and the application programming interface is implemented in accordance with the JavaScript language.
Claim 12	Claim 24
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.	The system of claim 13, 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.

I. MANDATORY NOTICES

A. Real Party in Interest

Petitioner Carvana, LLC (“Carvana”) is the real party in interest.

B. Related Matters

International Business Machines Corp. (“IBM”) asserted the ’719 Patent on April 3, 2024, in a pending action, *Carvana, LLC v. International Business Machines Corp.*, Case No. 1:23-cv-8616-KMK-VR (S.D.N.Y.) (D.I. 63) (“Related Litigation”).

IBM also asserted the ’719 Patent against a gaming company in *International Business Machines Corp. v. Zynga Inc.*, Case No. 1:22-cv-00590-GBW (D. Del.).

C. Lead and Back-up Counsel

Lead Counsel	Backup Counsel
Charley F. Brown Reg. No. 52,658 browncf@ballardspahr.com Ballard Spahr LLP 999 Peachtree Street, NE, Suite 1600 Atlanta, GA 30309	Additional Counsel Brian W. LaCorte will appear <i>Pro Hac Vice</i> Samhitha Medatia Reg. No. 70,996 medatias@ballardspahr.com Ballard Spahr LLP 999 Peachtree Street, NE, Suite 1600 Atlanta, GA 30309 Drew Hensley Reg. No. 79,305 hensleyd@ballardspahr.com Ballard Spahr LLP 1 East Washington Street, Suite 2300 Phoenix, AZ 85004

D. Service Information

Please address all correspondence to the addresses noted above. Carvana consents to electronic service by email to BallardCarvana719IPR@ballardspahr.com and the email addresses above.

II. GROUNDS AND REQUESTED RELIEF

Carvana petitions for *inter partes* review of claims 1–26 (the “Challenged Claims”) of the ’719 Patent and cancellation of the Challenged Claims as unpatentable based on the following grounds:

Ground	Claim(s)	Reference(s)	Basis
1	1, 13, 25, 26	Roy	§102
2	1, 13, 25, 26	Roy	§103
3	2, 4–12, 14, 16–24	Roy, Fennell, and O'Reilly	§103
4	3, 15	Roy and Fennell	§103

III. STANDING

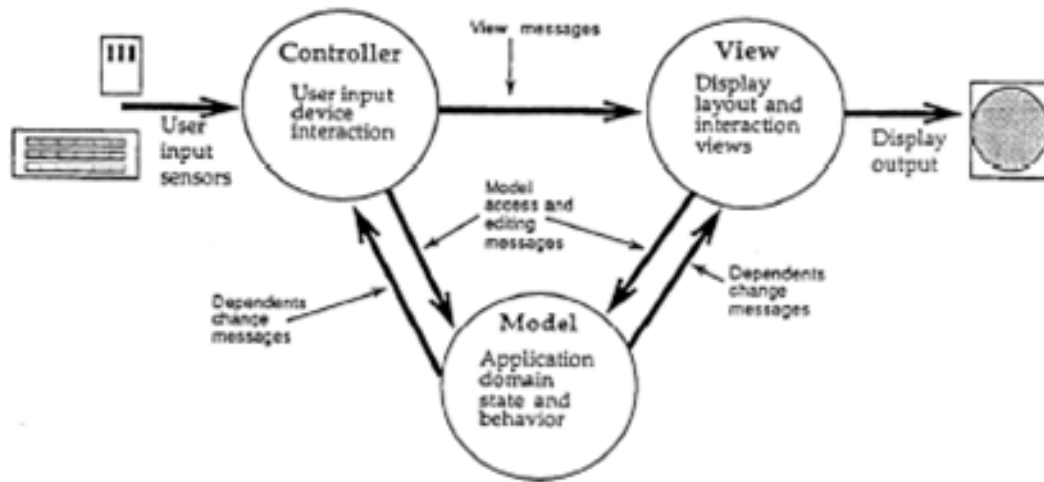
Carvana certifies that the '719 Patent is eligible for *inter partes* review and that Carvana is not barred or estopped from requesting IPR on these grounds.

IV. TECHNOLOGY

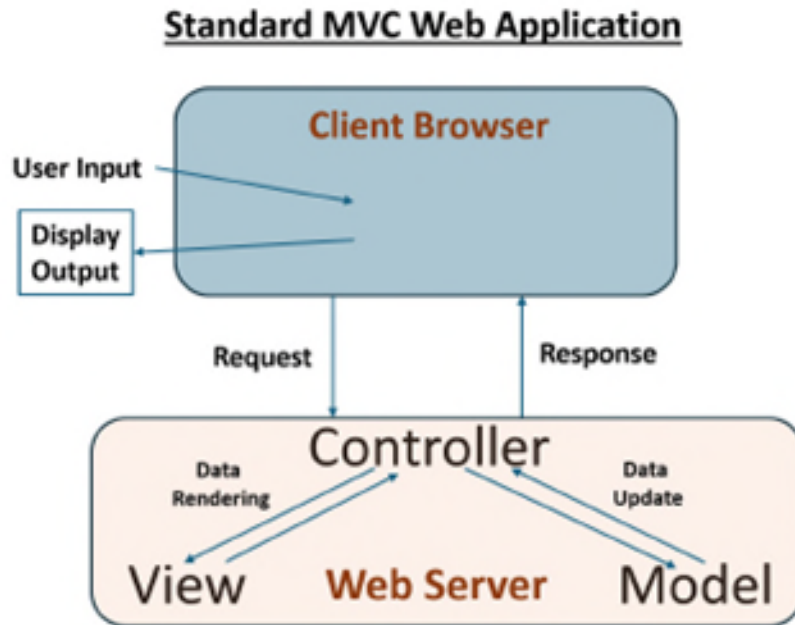
A. The Aged Technology of MVC and “Dual-MVC”

Model-View-Controller (MVC) is a nearly forty-year-old object-oriented user interface paradigm that divides interfaces into three related components: Model, View, and Controller. Ex.1003¶38. MVC was essential to an object-oriented programming language called Smalltalk-80 from the early 1980s.¹ Ex.1003¶39; Ex.1008 (Krasner). In the well-known MVC design, the *Model* maintains the state of an application; the *View* creates a representation of that state for viewing; and the *Controller* manages user interaction between the Model and View, Ex.1003¶40, as shown below (Ex.1008, 5):

¹ Smalltalk-80 applications typically had a single Model and several Views, and its display output was managed by a View. Ex. 1003¶43.



Ex.1003¶43. For example, a Model is a collection of data items (e.g., a list of library books); a View creates representations of such data (e.g., a sort of the books by author, title, or subject); and a Controller provides mechanisms for manipulating the model (e.g., process to check out a book). Ex.1003¶42. MVC's simplicity, encapsulation, and modularity made it a popular choice for developers designing early web applications with the client browser serving as the user interface:



Ex.1003¶45.

As developers created mechanisms for embedding code within MVC-based browsers (e.g., JavaScript) in the 1990s, developers divided MVC responsibilities between server and client. Ex.1003¶46. “Dual-MVC,” as it is called in the ’719 Patent, became widespread with the advent of web application environments facilitating such development. Ex.1003¶48. Indeed, any distributed implementation of Smalltalk-80 necessarily used “dual-MVC,” because an interactive application running it would have to have model, view, and controller components on every machine running the application. Ex.1003¶47; Ex.1009 (Bennett); Ex.1010 (Decouchant); Ex.1010 (McCullough). Other references predating the ’719 Patent, including U.S. Patent 5,838,909 (Roy), discussed below at §IV.D.1, disclose “dual-MVC” distributed applications. Ex.1003¶47. Having model, view, and controller

elements on both the client and server was not inventive at the time the '719 Patent was filed; instead a POSITA would have considered the concept routine at that time. Ex.1003¶47.

B. JavaScript Was Known at Least as of 1995

During the 1990s, developers looked for ways to create dynamic components on the client after a web page was loaded into the browser. Ex.1003¶49. This desire to add dynamic components to static HTML text led to the development of JavaScript, first released in 1995. Ex.1003¶¶49–50. JavaScript enabled web developers to create web pages with client-side interactivity, and when used with HTML, to create dynamic webpages. Ex.1003¶50. Using JavaScript, developers could create dynamic browser programs with fewer server interactions than traditional CGI methods. Ex.1003¶49.

By 1996, JavaScript communicated with the Java interpreter built into the browser and worked with and could control Java applets running in a web page. Ex.1003¶51. Although JavaScript and Java were distinct programming languages, they were said to make a “good team.” Ex.1003¶51; Ex.1006, 17–18 (§1.2.1).²

² Pincites correspond to Ex.1006's pagination.

Despite being well-known technology, the '719 Patent states that its claimed “dual-MVC” approach preferably uses JavaScript and HTML constructs. Ex.1001, 3:10–12.

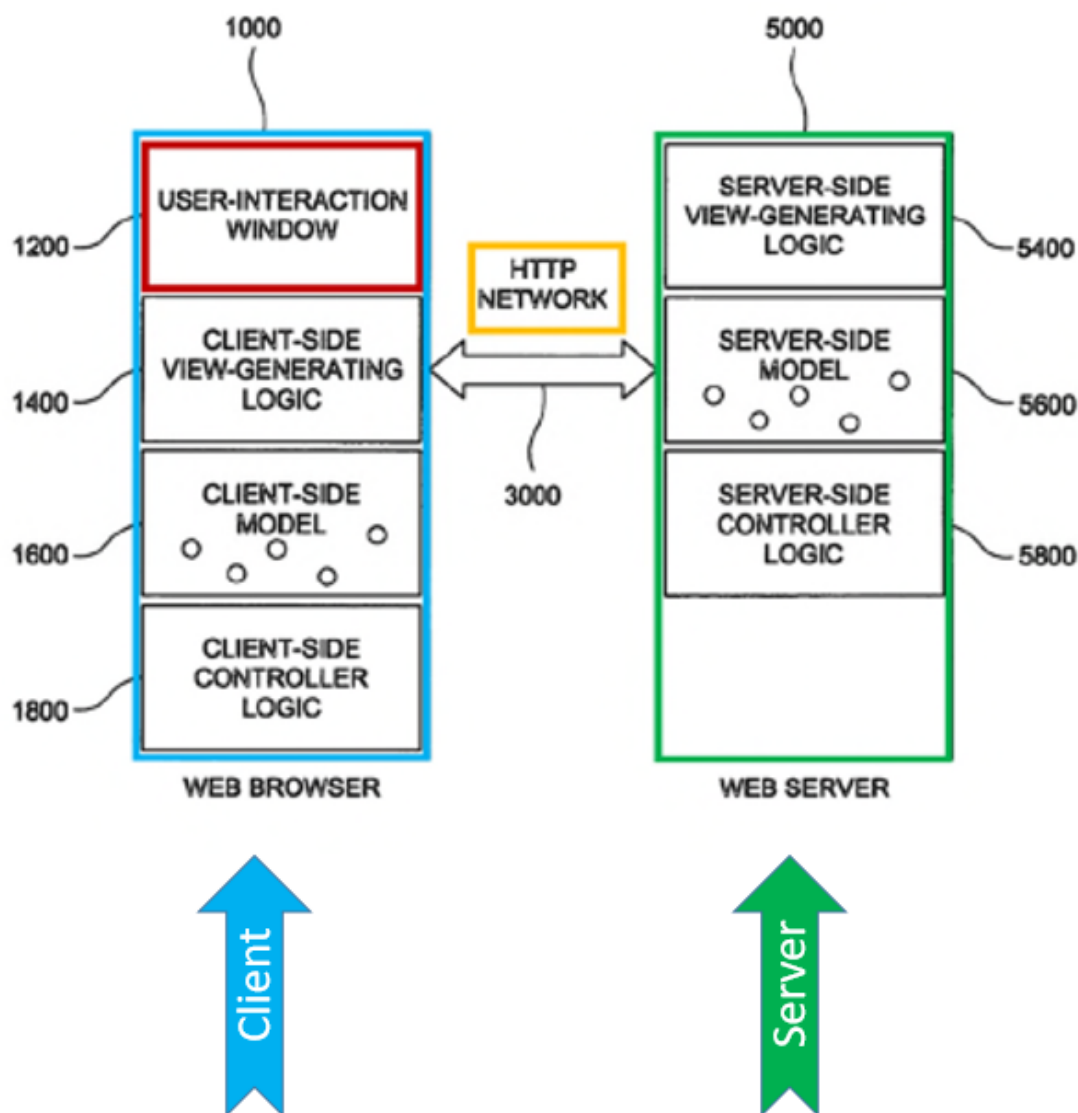
C. The '719 Patent Claims the Well-Known “Dual-MVC” Paradigm

The '719 Patent claims a “dual-MVC” web-based application covering a “dual-MVC” paradigm that was already well known, along with standard Internet and networking concepts of distributed applications, HTTP, HTML, and JavaScript as claimed in the dependent claims. Ex.1003¶¶53, 110.

1. Overview

The now-expired '719 Patent³ states a central objective of “eliminating many round-trips to the server.” Ex.1001, Abstract. It states that having a model, view-generating logic, and controller logic on both the client and server (which the '719 Patent calls “dual-MVC”) reduces server interactions. Ex.1001, 2:39–49; Ex.1003¶52. But this concept was not novel, because art predating the '719 Patent, such as Roy (discussed below), disclosed it. Ex.1003¶53. The '719 Patent depicts this “dual-MVC” paradigm as follows:

³ IPRs may be directed to expired patents. *Apple, Inc. v. Gesture Tech. Partners, LLC*, IPR2021-00922, Paper 10 (PTAB Nov. 29, 2021).



Ex.1001, Fig. 3 (annotated); Ex.1003¶54. The “User-Interaction Window” 1200 runs within a standard web browser, but the other MVC components are found on both client (blue box) and server (green box). “HTTP Network” 3000 is a standard HTTP network. Server-side MVC components (5000) are characterized as

“authoritative” copies; while client-side versions of these components (1000) are purportedly subsets of the server version. Ex.1001, 2:43–46; 5:14–27; Ex.1003¶55.

The ’719 Patent admits that Java applets already solved its stated problem of reducing client/server interaction. Ex.1001, 1:60–65. And though the ’719 Patent claims applets were larger than JavaScript MVC code, Ex.1001, 3:1–13, the patent concedes “JavaScript can be used to implement much of the same functionality that applets provide[.]” Ex.1001, 2:22–23; Ex.1003¶56.

The ’719 Patent states that when using “dual-MVC,” “the client machine maintains part of the Model, eliminating many round-trips to the server” because the client has stored some model data locally—a well-known concept. Ex.1001, 3:14–25; Ex.1003¶57.

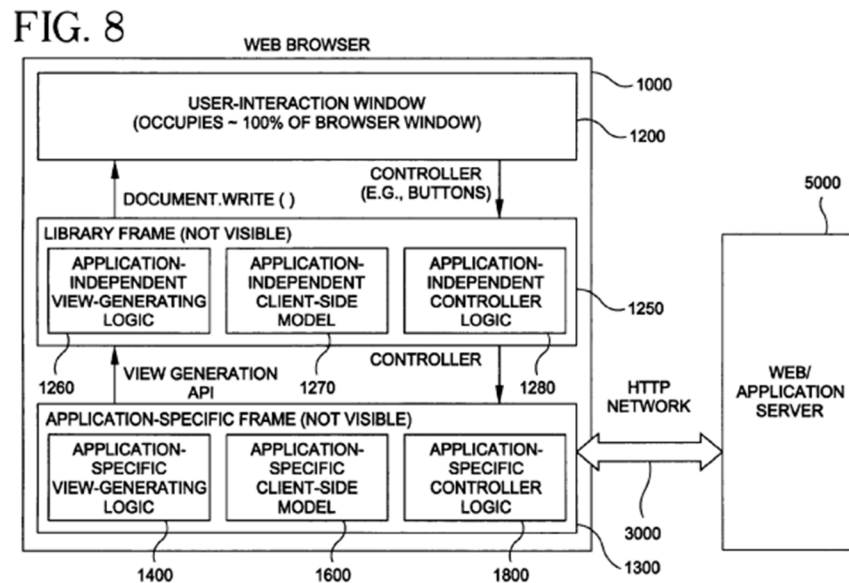
The ’719 Patent further claims HTML frames/framesets to implement “dual-MVC,” i.e., storing a portion of the model, view-generating logic, and controller logic on both server and client. Ex.1001, 3:26–32. But HTML frames/framesets were also well known before the ’719 Patent was filed. Ex.1003¶58.

Before the ’719 Patent was filed, developers commonly embedded JavaScript code on “invisible frames” (frames with locations outside the browser window or dimensions small enough to make them not visible on a screen) to update the visible frames. Ex.1003¶59. This technique avoided overwriting application code when the visible frame was modified or reloaded. Ex.1003¶59.

Invisible frames were likewise well known at the time the '719 Patent was filed.

Ex.1003¶60; Ex.1006, 208–09 (§12.4).

The patent further describes and claims a “not visible” “library frame” that employs a JavaScript API to generate an HTML document.



Ex.1001, Fig. 8, 7:51–58; Ex.1003¶61. The '719 Patent states this non-visible library frame contained “application-independent” MVC components, and provides examples of the “application-independent client-side model” as “caching and data-movement operations,” and “application independent controller logic” as “generic user-interface event handling.” Ex.1001, 7:51–58; Ex.1003¶62–63. The '719 Patent’s example of “application independent view-generating logic” is “logic to generate a button on the screen.” Ex.1001, 8:44–49; Ex.1003¶64. But caching

and data-movement operations, user-interface event handling, and placing user interface components within a web browser screen, were well known and practiced long before the '719 Patent was filed. Ex.1003¶64.

The '719 Patent contains 4 independent claims (claims 1, 13, 25, and 26), which overlap in many respects. Ex.1003¶65. Claims 2–12 and 14–24 ultimately depend from claims 1 and 13, respectively, and each set of dependent claims mirror each other. The '719 Patent claims the well-known concepts of storing the model, view-generating logic, and controller logic on both the server and client, via well-known means to do so, including JavaScript, HTML, and invisible frames. Ex.1003¶65.

2. Level of Ordinary Skill in the Art

A person of ordinary skill in art at the relevant time (“POSITA”) would have had at least a bachelor’s degree in computer science, computer engineering, or a related field, with at least two years of professional experience in the fields of networking and network-based systems or applications, such as client-server and web-based systems, or an equivalent level of skill, knowledge, and experience. Additional relevant work experience would substitute for education, and additional relevant educational experience would substitute for work experience. Ex.1003¶36.

3. Claim Construction

The parties in the Related Litigation have agreed to certain constructions of terms/phrases in the '719 Patent. Ex.1003¶17.

Claim Term(s) or Phrases(s)	Agreed Construction(s) for the Asserted Claims in the Related Litigation⁴	Relevant Claims in the '719 Patent
“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”	Preamble is limiting	1, 25, 26
“view generating . . . logic associated with the application”	program code that generates a screen or window representation of a subset of the model that the application chooses to display	1, 5, 12, 13, 17, 24–26
“the client and the server both locally . . . execute the view-generating and controller logic”	“view-generating logic” and “controller logic” executed on client must be the same “view-generating logic” and “controller logic” executed on the server, or some subset thereof	1, 13, 25, 26

⁴ In the Related Litigation, only claims 1–4, 8, 10–12, and 26 were asserted (“the Asserted Claims”). See *Related Litigation*, D.I. 105 (Joint Claim Terms Chart); D.I. 135 (Amendment to Joint Claim Terms Chart).

While the parties in the Related Litigation disagree on the constructions of the claim terms below, the '719 Patent claims are invalid under either set of the proposed constructions. Ex.1003¶18.

Claim Term(s) or Phrases(s)	Carvana's Proposed Constructions	IBM's Proposed Constructions Relevant to the Asserted Claims in the Related Litigation	Relevant Claims in the '719 Patent
application	"software that allows users to access and update data"	no construction necessary/plain and ordinary meaning	1, 3, 5, 8–13, 15, 17, 20–26
"model associated with [an/the] application"	"application data, rules, and algorithms affecting the application data"	"the data, rules, and algorithms affecting the data associated with the application" ⁵	1, 13, 25, 26
"controller logic associated with the application"	"program code that processes user requests and causes the model to be changed and the view to be refreshed"	"program code that processes user requests and causes the model to be changed and/or the view to be refreshed"	1, 13, 25, 26
frame(s)	"section(s) of a webpage, each constituting a distinct HTML document"	no construction necessary/plain and ordinary meaning	4–10, 16–22

⁵ This construction was adopted by the district court in *International Business Machines Corp. v. Zynga Inc.*, Case No. 22-590-GBW, D.I. 172 at 3 (D. Del. Oct. 30, 2023).

D. Prior Art

1. U.S. Patent No. 5,838,909 (Roy)

a. Roy Is Analogous Prior Art

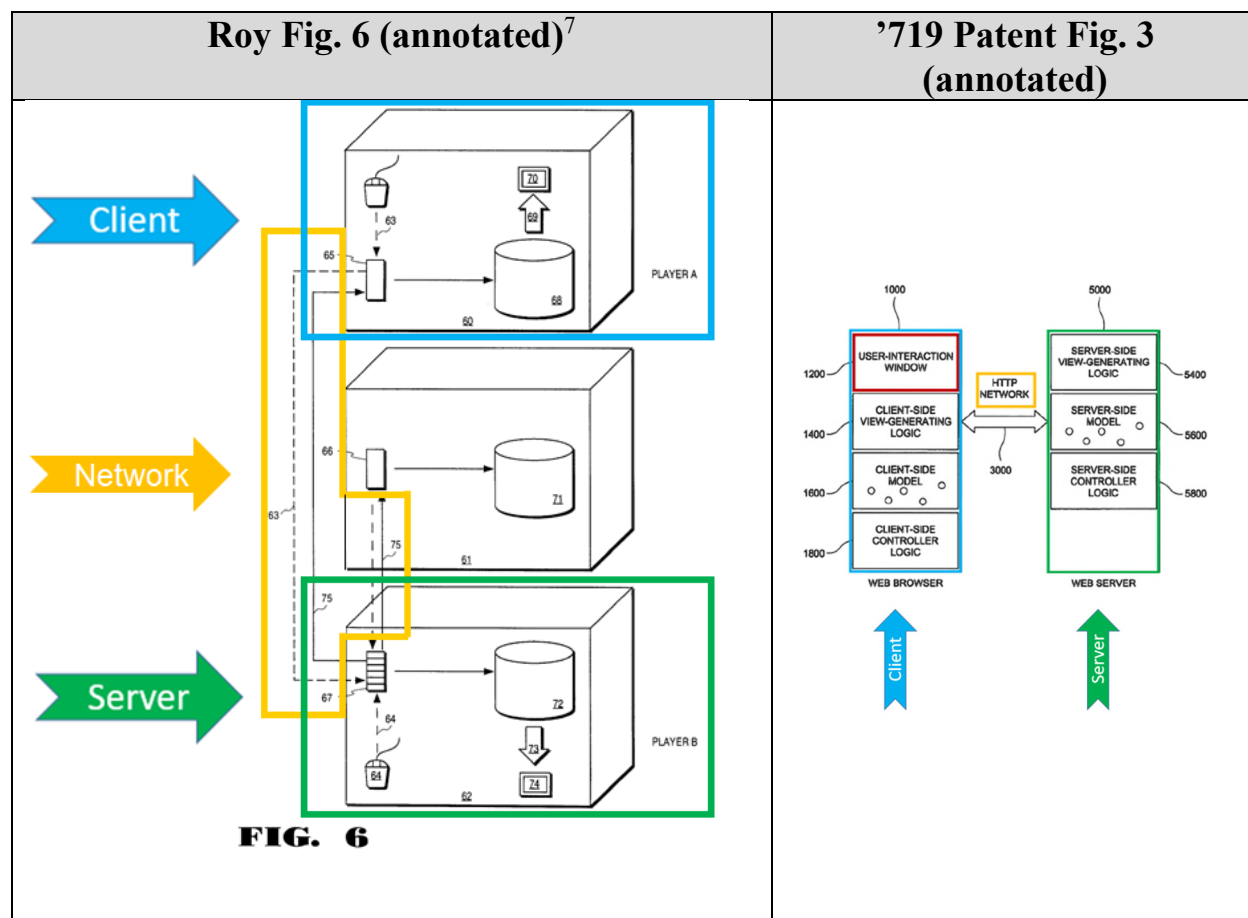
Roy discloses an Internet-based application that limits server roundtrips by storing a model, view-generating logic, and controller logic on both client and server, or “dual-MVC” application. Ex.1004. Roy, entitled “Reducing Latency When Synchronizing Access to a Multi-User Database Over a Network,” was filed May 23, 1996. Ex.1004. It is neither identified on the face of the ’719 Patent, nor was it considered during prosecution. Ex.1001; Ex.1002; Ex.1003¶79.

A POSITA seeking to reduce client/server interactions would look to Roy because it too uses “dual-MVC” to solve the problem of reducing latency when synchronizing access to a networked database.⁶ Ex.1003¶80; Ex.1004, 1:5–9. The ’719 Patent describes a need for Internet-based “applications which [do] not require continual network communication between the client and server.” Ex.1001, 1:48–2:35; Ex.1003¶80. Roy provides the very solution to this problem. Roy’s system reduces client/server interactions by enabling both server and client to process events locally whenever possible. Ex.1003¶81.

⁶ Art is analogous when it is (1) within the field of the inventor’s endeavor or (2) reasonable pertinent to the particular problem with which the inventor was involved. *Ethicon LLC v. Intuitive Surgical, Inc.*, No. 2021-1601, 2022 WL 1576779, at *3 (Fed. Cir. 2022) (nonprecedential). Roy is both.

b. Roy Discloses a “Dual-MVC” Networked System

Roy discloses Internet-connected computers with stored application software, like a simulated American football computer game. Ex.1004, Fig. 1, 3:23–35; 3:48–56; Ex.1003¶82. In the game, players controlling the offense need to control factors like speed and direction, whereas players controlling the defense only need to select predefined defensive patterns. Ex.1004, 3:23–35; Ex.1003¶82. The player on offense therefore has a more restrictive latency requirement. Ex.1004, 3:23–35; Ex.1003¶82. Roy recognizes the computer employed by the user with the most restrictive latency requirement could serve as the “master event server” (i.e., **server**) in the system, locally updating the game database and thus reducing the effects of latency. Ex.1003¶82–83. Other players are “slaves,” (i.e., **clients**) able to locally play the game with minimal server interaction. Ex.1003¶82–83; Ex.1004, 4:37–43.



Like the dual-MVC **client** and **server** in '719 Patent Figure 3, Roy Figure 6 shows a **client** and **server**. In Roy, the computer system containing the master event server and master database is the “server;” the other computers are clients. Ex.1004, Fig. 7–9C (flowcharts referring to “server” of the system); Ex.1003¶84–85. Further, like the '719 Patent’s description of a model, view-generating logic, and controller logic existing on both **client** and **server**, Roy likewise describes: a

⁷ **Green** components correspond to the **server**; those in **blue** correspond to the **client**. Yellow indicates the network connection.

(1) **model** in client-side **slave copy database 68** and server-side **database 72**;
(2) **view-generating logic** in client-side **rendering commands 69** and **display 70**
and server-side **rendering commands 73** and **display 74**; and (3) **controller logic** in
client-side generated **events 63** and **event server 65** and server-side generated
events 64 and **event server 67**. Ex.1003¶85; Ex.1004, Fig.6, 8:60–9:5.⁸

Player A in Figure 6 demonstrates the client side of Roy’s dual-MVC architecture. Player A can locally play the game with minimal server interaction because Player A’s **rendering commands 69** generate **display 70** of the game (the “view” of the dual-MVC) from the **slave copy 68** of the master database (the “model” of the dual-MVC), rather than the master database directly. Ex.1004, 5:33–35.⁹ Updating the local view from local rendering commands reduces bandwidth consumption on the network. Ex.1004, 5:29–33; Ex.1003¶86.

The client controller logic (“control” of “dual-MVC”) handles Player A’s input **events 63** (such as mouse button clicks). **Slave event server 65** receives Player A’s input and then based on a determination from local controller logic,

⁸ While one embodiment of Roy described in Fig.5A uses computer 61 “initially” as a “central server,” in Fig.6 the master event server and master database (i.e., the “server”) have migrated to the most active user, and computer 62 is now the master server. Ex.1003¶85n.54; Ex.1001, Fig.5A, 5:17–36; 8:60–9:5. For purposes of comparing the claim elements of the ’719 Patent to Fig.6, **computer 62** is the “server,” and **computer 60** is the “client.” Ex.1003¶85n.54.

⁹ While this description references Fig.5A, 5:17–36, it also applies to Fig.6.

either **retains and handles the events locally**, or passes them along to the **master event server 67**. Ex.1004, 5:43–46; 8:60–9:5 (noting that in Fig.6, **the master event server is 67**); Ex.1003¶87.

Roy's handling of events locally minimizes server interaction, the goal noted and claimed in the independent '719 Patent claims. Locally retained events are those “purely local” events which set display options or other functions relevant only to Player A's computer. Ex.1003¶88; Ex.1004, 5:40–43. Non-local events, i.e., events received by the **master event server**, are used to update the **master database**. Ex.1004, 5:37–49. This architecture reduces server interaction—the same goal the '719 Patent describes and claims—because not all events travel to the master server. Ex.1003¶88.

Player B in Figure 6 demonstrates the server side of Roy's “dual-MVC” architecture. While playing, Player B's **rendering commands 73** generate **display 74** of the game (“view” of the “dual-MVC”) from **master database 72** (“model” of the “dual-MVC”).¹⁰ Ex.1003¶89.

The server controller logic (“control” of the “dual-MVC”) handles Player B's input **events 64** (such as mouse button clicks). **Master event server 67** receives

¹⁰ In Fig.6, **database 72** is the master database. Ex.1004, 8:52–9:5 (referencing Fig.6 after Player B hosts the master event server); 4:44–55 (noting the single master database in the system is preferably on the same system as the master event server); 5:29–33 (referring to Fig.5A embodiment).

Player B's input and then passes the events to [slave event servers](#). Ex.1004, 5:46–49; Ex.1003¶90.

The '719 Patent provides a similar example of a “dual-MVC” networked system in the form of an online “help desk.” Ex.1001, 6:1–45. In the '719 Patent's help-desk program, a “consultant” assisting a “customer” opens an interface and asks for information like ID, operating system, and hostname. Ex.1001, 6:4–31. In the prior art embodiment, each request required a round trip to the server to populate corresponding data. Ex.1001, 6:33–35. The '719 Patent's purported improvement was storing more data on the consultant computer (client) to eliminate server roundtrips. Ex.1001, 6:35–45 (explaining when the client stores a subset of the model, server interactions “can be eliminated, thus providing better performance”). This is precisely what Roy discloses. In Roy, each client (e.g., Player A in Figure 6) has a copy of the model ([slave copy of the master database 68](#)), eliminating server roundtrips. Ex.1004, Fig.6; Ex.1003¶91.

Thus, Roy teaches a client/server system each with a **model**, **view-generating logic**, and **controller logic**, consistent with the '719 Patent's “dual-MVC” disclosure. Ex.1003¶92.

2. U.S. Patent No. 5,695,400 (Fennell)

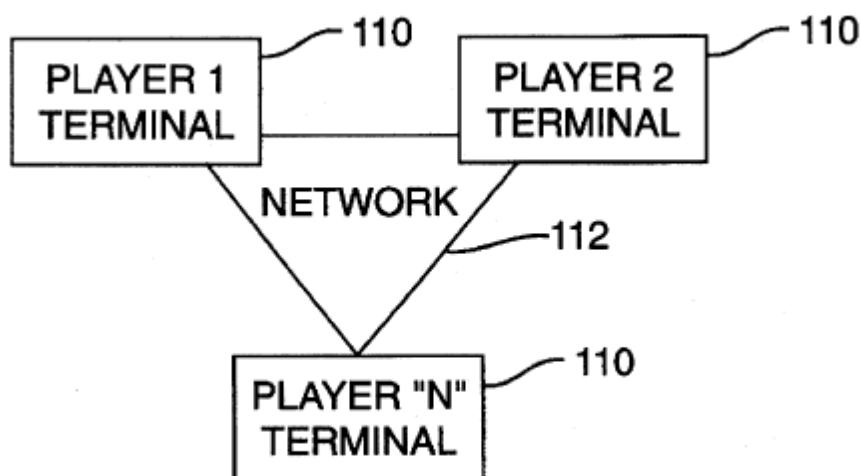
a. Fennell Is Analogous Prior Art

Fennell, like Roy, describes playing multiplayer games over the Internet. Fennell, entitled “Method of Managing Multi-Player Game Playing Over a Network,” Ex.1005, was filed January 30, 1996. It is neither identified on the face of the ’719 Patent, nor was it considered during prosecution. Ex.1003¶93.

Fennell stands in the ’719 Patent’s field of endeavor as both disclose an Internet-based system that reduces host-computer interactions, contemplates differences in connection speed between the client and server and “compensates for network latency among different terminals.” Ex.1001, 3:6–10, Ex.1005, Abstract, 1:39–43; 3:1–2. As Fennell recognizes, network latency effects make a game “less enjoyable and therefore less popular,” and thus Fennell’s invention minimizes the effects of network latency. Ex.1005, 1:25–36; Ex.1003¶94.

While traditional gameplay requires interaction with a host computer, Fennell discloses an embodiment minimizing host-computer interaction by equipping each user with a model (game challenge), view (display) and controller (terminal equipped with an input/output device). Ex.1005, 5:38–53; Ex.1003¶95–96. In that embodiment, terminal 110 downloads an encrypted question-and-answer file (the model) and causes that model to be sent to additional participants. Ex.1003¶97. The additional participants do not interact with the host computer, but

rather only terminal 110, via network 112. Each additional participant therefore has a copy of the model, a display (view) and controller (terminal equipped with “any suitable input/output device”), the same “dual-MVC” architecture the '719 Patent discloses:

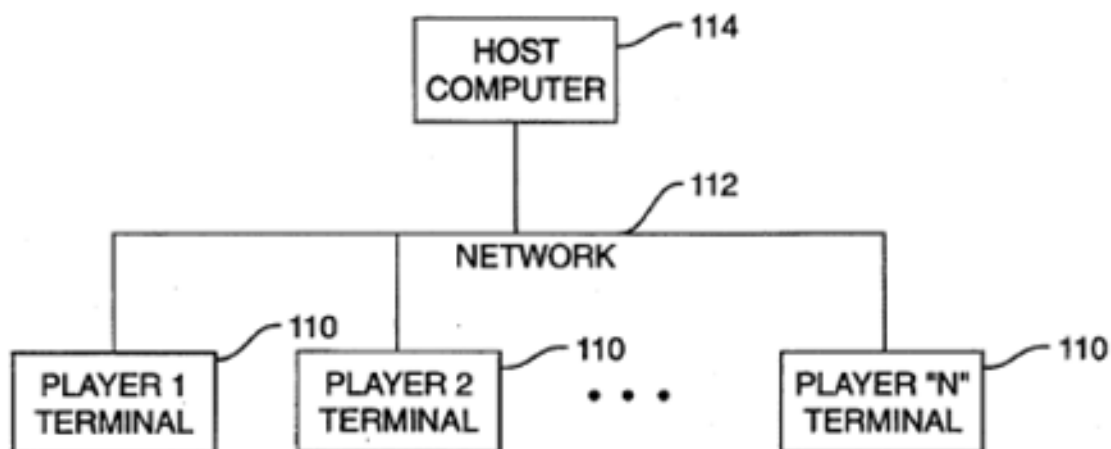


Ex.1005, Fig. 3. Accordingly, like the '719 Patent and Roy, Fennell discloses a distributed system with each terminal having a model, view, and controller.

Ex.1003¶98.

b. Fennell Relates to Roy and Discloses Distributed Internet Applications Accessed Via Browser Software

Figure 1 of Fennell describes a host computer and plurality of network-connected terminals:



The Fennell “terminal” refers to any suitable input/output device such as a PDA or teletype, Ex.1005, 3:3–8, but Fennell also teaches the terminal could be “a smart terminal such as a World Wide Web browser with software ‘applets’ (written in, e.g., Sun Microsystems’ Java language) downloaded on demand,¹¹ or a general-purpose computing device with suitable game programming that is conventionally loaded from a hard disk, CD-ROM, network server, or other mass storage device when the game is to be played.” Ex.1005, 3:11–19. Indeed, Fennell suggests that many embodiments would be “a dedicated device” like “World Wide Web browser with software ‘applets,’” given the low marginal cost of distributing software,

¹¹ At the time of the ’719 Patent, it was known that Java applets could be controlled by JavaScript. Ex.1003¶101.

which would inform a POSITA seeking to distribute Roy's system. Ex.1005, 3:19–22; Ex.1003¶100.

A POSITA would appreciate that Fennell's disclosure of a browser-based application over the Internet necessarily discloses using HTML. Ex.1003¶101.

Accordingly, Fennell discloses a browser-enabled online application that minimized client/server interaction and, as noted above, discloses an embodiment that, like Roy's system, allowed users to serve as the "host," minimizing host-computer/server interaction. Ex.1003¶102.

3. *JavaScript: The Definitive Guide* (O'Reilly)

a. O'Reilly Is a Printed Publication and Analogous Prior Art

Though O'Reilly was published in January 1997, it is neither identified on the face of the '719 Patent, nor was it considered during prosecution, despite the inventors being aware of the series. Ex.1003¶¶103, 109; Ex.1006; Ex.1014 (citing JAVASCRIPT: THE DEFINITIVE GUIDE (Flanagan, David) (O'Reilly, pub.) (3d. Ed. 1998)).

O'Reilly is a printed publication. In *Hulu, LLC v. Sound View Innovations, LLC*, IPR2018-01039, Paper 29 (PTAB Dec. 20, 2019), the Board found another book in the O'Reilly series was a printed publication, and characterized it as "a textbook from an established publisher," and a "well-known book series." A POSITA at the time of the '719 Patent would have likely read and consulted

O'Reilly. Ex.1003¶¶104–107. Indeed, the '719 Patent cites another book in the O'Reilly series, *Dynamic HTML, The Definitive Reference*, and incorporates it by reference. Ex.1001, 2:15–20; Ex.1003¶108.

The '719 Patent is further directed to using then-known concepts such as HTTP (e.g., Claims 2 and 14), frames (e.g., Claims 4–6, 8–10, 16–18, 20–22), invisible frames (e.g., Claims 7 and 19), HTML and JavaScript (e.g., Claims 11 and 23), API controls (e.g., Claims 10 and 22), and application independent features including caching, view-generating logic, and controller logic (e.g., Claims 8–10, 20–22). O'Reilly discloses all of these, making it analogous art. Ex.1003¶110.

b. O'Reilly Demonstrates Familiarity with HTTP as of at Least 1997

Claims 2 and 14 of the '719 Patent claim communication over HTTP. But O'Reilly demonstrates that by 1997—years before the '719 Patent was filed—readers were already familiar with HTTP networks. Ex.1006, 21 (§1.3.5); Ex.1003¶111.

When discussing Server-Side JavaScript implementations, O'Reilly explains how to handle certain HTTP requests, Ex.1006, 32 (§1.6.5), and implement cookies as “an extension to the HTTP protocol.” Ex.1006, 234 (§15.1);

Ex.1003¶¶111. O'Reilly demonstrates that HTTP was well known by at least 1997.

Ex.1003¶111.

c. O'Reilly Discloses Using Frames and Invisible Frames to Generate HTML Documents and Associate View-Generating and Controller Logic, the Same Subject Matter Purported to Be (But Was Not) Invented in the '719 Patent

Claims 4–10 and 16–22 of the '719 Patent claim frames and invisible frames, as well as view-generating logic and controller logic, suggesting that such use was novel. But O'Reilly describes the well-known concept of using JavaScript to create views and control frames in HTML documents years earlier.

Ex.1003¶112.

O'Reilly explains that browsers could display “multiple pages within a single window by using *frames*—a feature that allows a single browser window to be divided horizontally or vertically (or both) into individual sections that each display a separate HTML document.” Ex.1006, 184 (§11.3) (emphasis in original);

Ex.1003¶¶113. O'Reilly explains that at that time, it was “much more common to create a web site that uses multiple frames than it [was] to create one that uses multiple browser windows.” Ex.1006, 207 (§12.4).

O'Reilly also teaches the use of **invisible** frames to dynamically update HTML documents. Ex.1003¶114. It discloses a programmer could use JavaScript to create dynamic HTML documents in separate windows or frames, recognizing

the problem of using JavaScript to dynamically create an HTML document within its *own* window or frame, “because this generally overwrites the JavaScript code itself!” Ex.1006, 208 (§12.4).¹² O’Reilly even provides sample code to deploy this technique. Ex.1006, 208–09 (§12.4); Ex.1003¶¶115–117.

O’Reilly notes a developer can use JavaScript to generate documents and write arbitrary HTML into a document as the document is being parsed by the browser. Ex.1006, 19 (§1.3.1); Ex.1003¶118. O’Reilly teaches a programmer how to use JavaScript to specify view-related features of an HTML document, which it says works “particularly well in multiframe documents.” Ex.1006, 19 (§1.3.1).

O’Reilly also teaches a programmer to use both JavaScript and HTML when creating framed layouts. Ex.1006, 19 (§1.3.2). O’Reilly explains that JavaScript provides control over which web pages are displayed in particular frames of the browser. Ex.1006, 19 (§1.3.2); Ex.1003¶119.

In short, O’Reilly discloses the use of JavaScript with frames and invisible frames in the generation of HTML documents, the very subject matter claimed in the ’719 Patent.

¹² Ex.1001, 3:47–50 (“a client-side application attempting to update its View will destroy itself by overwriting its own Controller logic and Model data”).

d. O'Reilly Discloses the Use of APIs

The '719 Patent claims JavaScript APIs (e.g., Claims 10 and 22). But O'Reilly discloses the use of APIs to interact with software years before the '719 Patent was filed. Ex.1006, 307 (§19.6.3); Ex.1003¶120).

e. O'Reilly Discloses Using JavaScript to Create and Modify HTML Documents in Browsers, Including Interaction with Java—the Same Subject Matter Claimed in the '719 Patent

The '719 Patent claims the use of HTML and JavaScript together in claims 11 and 23, suggesting this combination was novel. But O'Reilly described this years earlier. Ex.1006, 4–9 (Table of Contents); Ex.1003¶121.

O'Reilly explains JavaScript allows a programmer to include executable content on web pages, such that a web page did not need to be a static HTML document, but rather could include dynamic programs that interacted with a user, controlled the browser, and dynamically created HTML content. Ex.1006, 14 (§1). O'Reilly explains a programmer can use JavaScript to interact with HTML form documents and control the behavior of those forms by defining “event handlers” that respond to, for example, button clicks. Ex.1006, 15 (§1.1). O'Reilly explains that the “interactions between JavaScript code and the web browser and the browser’s contents are what matter most” and that mastering the ability of JavaScript to interact with content on a web page and communicate with Java

applets and plug-ins was the real key to using JavaScript effectively. Ex.1006, 17 (§1.1); Ex.1003¶122.

Despite the '719 Patent's attempts to distinguish its claimed use of JavaScript from prior art Java applets, it admits that "JavaScript can be used to implement much of the same functionality that applets provide." Ex.1001, 2:22–23. O'Reilly teaches how to translate between Java applets and JavaScript. O'Reilly explains that JavaScript communicates with the Java interpreter built into the browser and can work with and control Java applets in a web page. O'Reilly notes that Java and JavaScript could both deliver "executable content" over networks and that JavaScript and Java "make a good team." Ex.1006, 17–18 (§1.2.1); Ex.1003¶123.

O'Reilly also explains that client-side JavaScript can preprocess "input data, which can reduce the amount of data that must be transmitted to the server" Ex.1006, 20 (§1.3.3) and that server-side JavaScript can "dynamically generate HTML to be displayed by the client." Ex.1006, 32 (§1.6.5); Ex.1003¶124.

f. O'Reilly Describes JavaScript's Caching Functions, the Same Functions Claimed as "Application-Independent" in the '719 Patent

While the '719 Patent discloses caching as an example of an application-independent model, O'Reilly described this years earlier. Ex.1001, 7:58–61; Ex.1003¶125.

O'Reilly explains that allowing JavaScript files to be cached by the browser allows them to load much more quickly, Ex.1006, 164–66 (§10.1.2, 10.2), and even teaches a method of caching images for easy replacement into dynamic HTML pages. Ex.1006, 242 (§16.1); Ex.1003¶¶126–127.

g. O'Reilly Discloses “Event Handlers” that Provide Controller Logic

Though the '719 Patent claims “controller logic,” O'Reilly discloses controller logic in the form of event handlers before the '719 Patent's filing date. Ex.1003¶128.

O'Reilly explains that a developer could use JavaScript to create event handlers (controller logic) to be used in client-side JavaScript. An event handler is a special-purpose function used only in client-side JavaScript that a programmer can associate with a button in an HTML form, for example, which could then trigger a piece of JavaScript code. Ex.1006, 108; 166–71 (§6.5; §10.3). O'Reilly characterizes event handlers as “almost always the central element of any interesting JavaScript program.” Ex.1006, 252 (§17.1); Ex.1003¶129.

In short, O'Reilly discloses the use of HTTP, frames, invisible frames, HTML, JavaScript, API controls, caching, view-generating logic, and controller logic—the same supposed inventive concepts claimed in the dependent claims of the '719 Patent. Ex.1003¶130.

V. THE '719 PATENT IS INVALID

A. Ground 1: Roy Anticipates Claims 1, 13, 25, and 26

Because Roy discloses every element of the '719 Patent independent claims, the independent claims are invalid. 35 U.S.C. §102; Ex.1003¶132.

1. Preamble¹³

Roy discloses “a method for use” (claims 1 and 26) and “an article of manufacture” (claim 25) in a “client/server system” (claims 1, 25 and 26) or “network system” (claim 13) (e.g., the system depicted in Figure 6) of “reducing interactions between a client and server” (claims 1, 25 and 26) (e.g., Roy discloses that “local events” are processed locally) “in association with an application being accessed by the client at the server” (e.g., a user connects to an online application). Ex.1003¶132.

First, Roy discloses a “client/server system,” which was also a “network-based system:” “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.” Ex.1004, 2:38–40; 2:31–32 (describing FIG.4B’s “client-server configuration”); Ex.1003¶134.

¹³ The Related Litigation parties agree the preamble is limiting as to claims 1 and 26.

Second, Roy discloses a method for “reducing interactions between a client and server in association with an application being accessed by the client” (relevant to claims 1, 25, and 26) in multiple examples:

(i) Roy explains how providing slave copies of the database to every non-master computer connected to the system reduces the need to send rendering commands across the network:

“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[.]”

Ex.1004, 4:48–54; Ex.1003¶¶135–136;

(ii) Roy discloses that a user creates an “event” each time a user issues a command and if the event is determined to be a local event, then it is processed by the client and the results displayed, thereby eliminating the need for server interaction. Ex.1003¶137;¹⁴ and

(iii) Roy’s Figure 6 embodiment shows Player B’s computer is the master server (“server” in ’719 Patent parlance) and *all* events generated by Player B are processed “locally.” Ex.1004, 6:5–14, Fig.6 (annotated); Ex.1003¶138.

¹⁴ Local events include events which set display options or other functions that are only relevant to that computer. Ex.1004, 5:40–43.

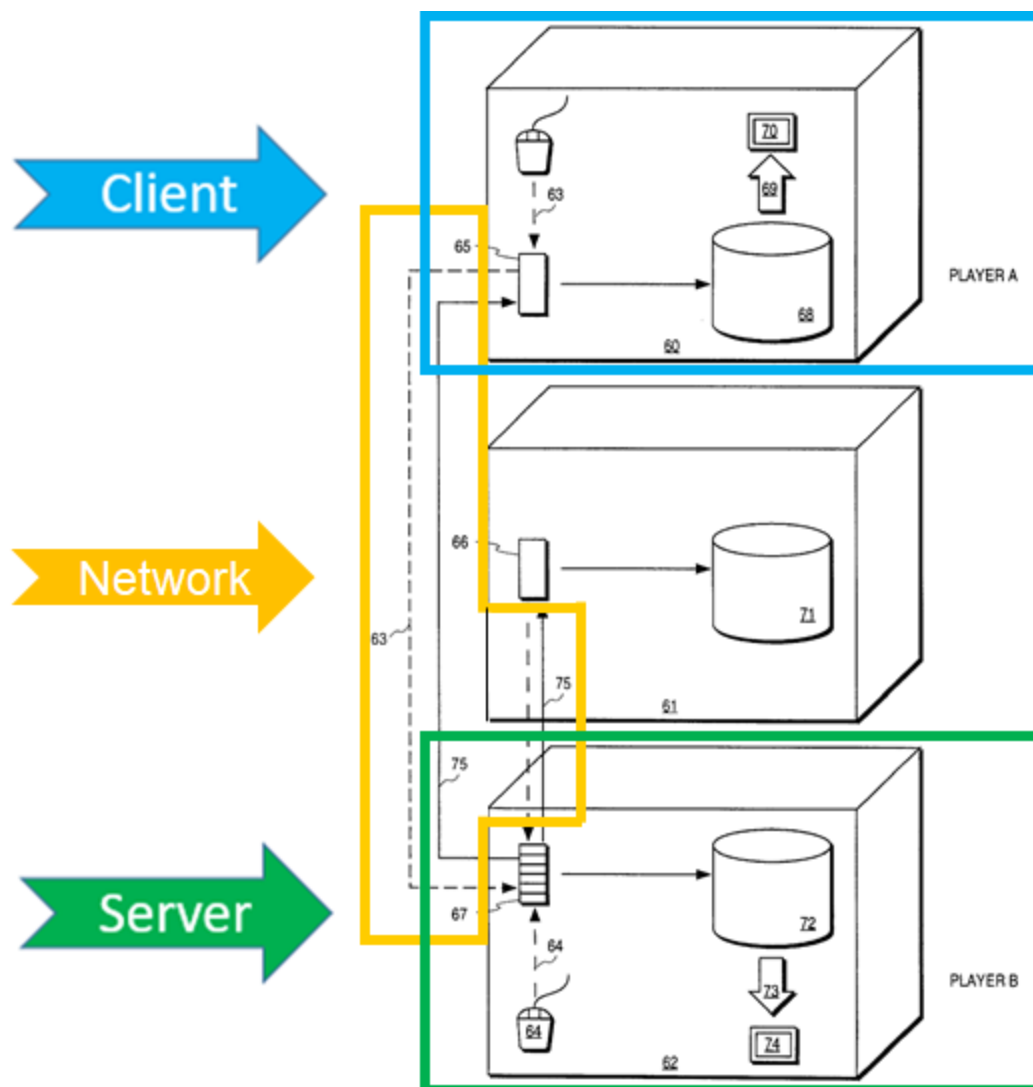


FIG. 6

Third, Roy discloses an “application being accessed by the client” (e.g., computer system 60) at the “server” (e.g., master event server 67 on computer system 62). In other words, “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.” Ex.1004, 2:38–41; Ex.1003¶139.

This was accomplished by “machine readable media containing one or more programs,” which, when executed, implement the application. Ex.1003¶140.

Therefore, Roy discloses the client accessing the application at the server.

Accordingly, Roy discloses each limitation of the preambles of claims 1, 13, 25, and 26. Ex.1003¶141.

2. Limitation (a)

Roy discloses independent claim limitation (a). Roy discloses “configuring the server” (e.g., components in **computer 62**) “to store a model associated with the application” (e.g., **database 72** with the world model for the application) “and to execute view-generating and controller logic associated with the application,” as those terms have been construed and proposed for construction. This would necessarily require a system in which the “server has at least one processor operative” to conduct these actions. Ex.1003¶142. Roy discloses these elements under either parties’ proposed claim constructions in the Related Litigation. Ex.1003¶142.

- configuring the server (e.g., **computer 62 with the master event server and master database**) to store the data, rules, and algorithms affecting the data associated with the application (e.g., **database 72** with the world model for the application) and application data, rules, and algorithms affecting the application data (e.g., **database 72** with the world model for the application);
- configuring the server (e.g., **computer 62 with the master event server and master database**) to execute program code that

processes user requests (e.g., **event server 67** processing **events 64**) and causes the model to be changed **and/or** the view to be refreshed (e.g., **database 72** with the world model for the application is changed and the **display 74** is updated by **rendering commands 73**); and

- configuring the server (e.g., **computer 62 with the master event server and master database**) to execute program code that generates a screen or window representation of a subset of the model that the application chooses to display (e.g., the server generates an updated **display 74** via **rendering commands 73**).

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.” Ex.1004, Abstract, 4:8–40. Roy further explains how the “game world can be modeled by a database” which can be located on a user’s computer. Ex.1004, 1:28–30. Thus, whichever construction is adopted for “the model associated with the application,” the “configuring the server” limitation is met because the database contains the model of the application required to operate the application software. Ex.1003¶143.

Second, Roy discloses configuring **the server** to “execute view-generating [] logic associated with the application.” For example, in Figure 6, **computer 62** is the server and contains the master event server and master database. Ex.1004, 6:1–11. It receives **events 64** from Player B on **computer 62** (and from other users), updates **master database 72**, and generates **rendering commands 73** that update the **display**

74. Ex.1004, Fig.6, 4:50–55, 7:1–8. This functionality, described in relation to Figure 5A, is illustrated in Figure 6 (annotated):

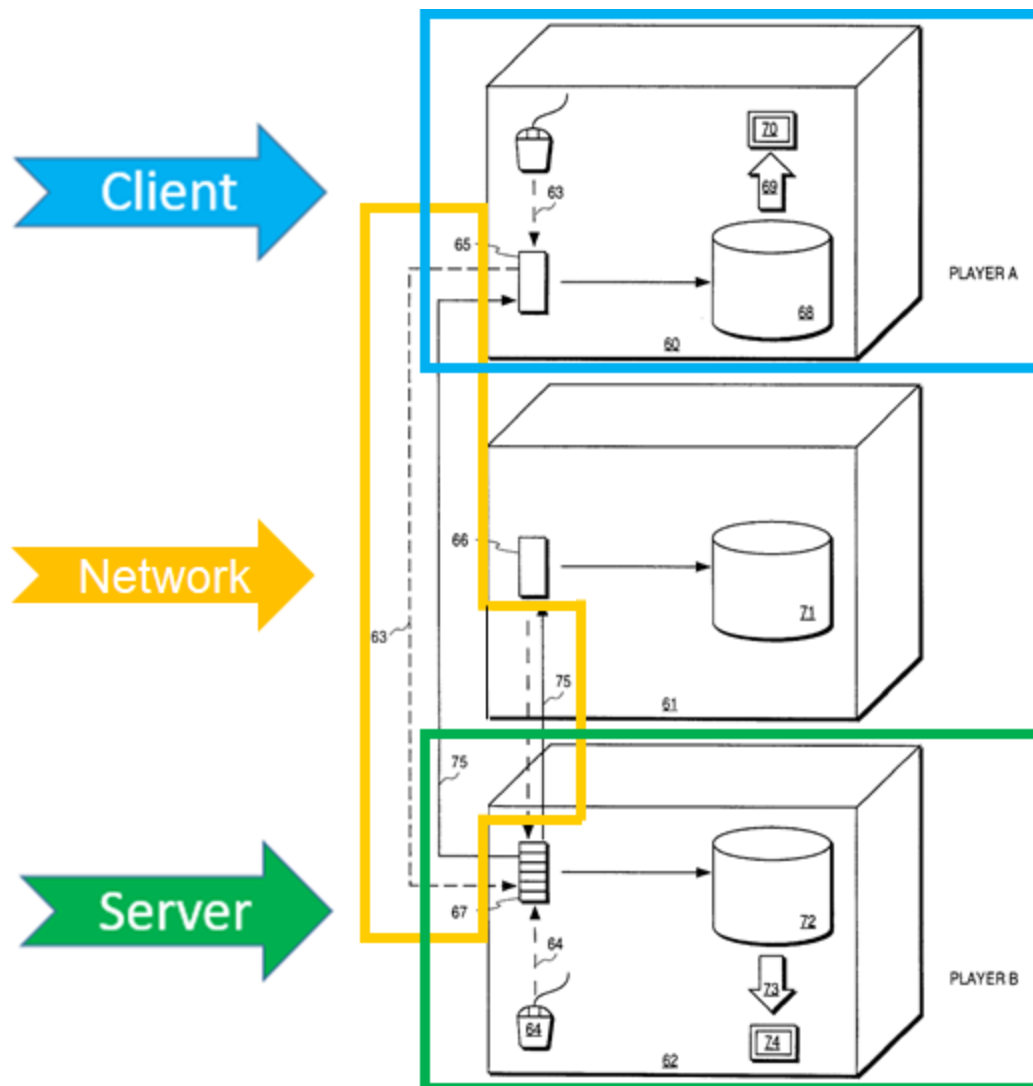


FIG. 6

Server computer 62 executes view-generating logic when it creates rendering commands 73 used to update the display 74 for Player B.

Ex.1003¶¶144–146.

Third, Roy discloses the server “execut[ing] [] 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 application model). Ex.1004, Abstract; Ex.1003¶147.

In Figure 6, **computer 62** as server receives **events 64** from Player B. Ex.1003¶148. Roy discloses the user can generate “events” using a mouse, joystick, trackball, keyboard, or any other I/O device. Ex.1004, 3:66–4:13; Ex.1003¶148. **Computer 62’s event server 67** then executes controller logic to update **master database 72** and causes **rendering commands 73** to be rendered and shown to the user on the user’s **display 74**. Ex.1004, Fig.6, 5:27–49; Ex.1003¶148. Because **computer 62** is the server in the client/server system, Roy discloses configuring the server to store and execute controller logic. Ex.1003¶148.

Accordingly, Roy discloses each element of limitation (a). Ex.1003¶149.

3. Limitation (b)

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., **slave copy of master database 68**) and “to execute at least a subset of the view-generating and controller logic associated with the application.” Ex.1003¶150. Roy discloses these

elements under either party's proposed claim constructions in the Related Litigation. Ex.1003¶150.

- 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., [slave copy of master database 68](#)) and application data, rules, and algorithms affecting the application data (e.g., [slave copy of master database 68](#));
- configuring the client (e.g., [computer system 60](#)) to execute program code that processes user requests (e.g., [event server 65](#) processing [events 63](#)) and causes the model to be changed and/or the view to be refreshed (e.g., [events 63](#) cause [slave copy of master database 68](#) to be changed and [display 70](#) is refreshed to reflect the change);
- configuring the client (e.g., [computer system 60](#)) to execute program code that generates a screen or window representation of a subset of the model that the application chooses to display (e.g., client [computer system 60's rendering commands 69](#) generate an updated [display 70](#)); and
- 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 network).

First, Roy discloses “configuring the client to store at least a subset of the model associated with the application.” In particular, Roy discloses “[s]lave copies of the master database (or portions thereof) are stored” on each client computer. Ex.1004, 4:44–48. The databases of Roy represent the world model for the application. Ex.1003¶151. Because the clients contain a copy of the master

database (or portion thereof), the client is “configured” “to store at least a subset of the model.” Ex.1003¶151.

Thus, as an example, the master database “or portions thereof” (Ex.1004, 4:47–48) 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.

Ex.1003¶152; Ex.1004, Fig.6. This limitation is therefore met whether a “model” is “application data, rules, and algorithms affecting the application data” or “the data, rules, and algorithms affecting the data associated with the application,” because the database contains the model of the application which is required to interact with the application. Ex.1003¶152.

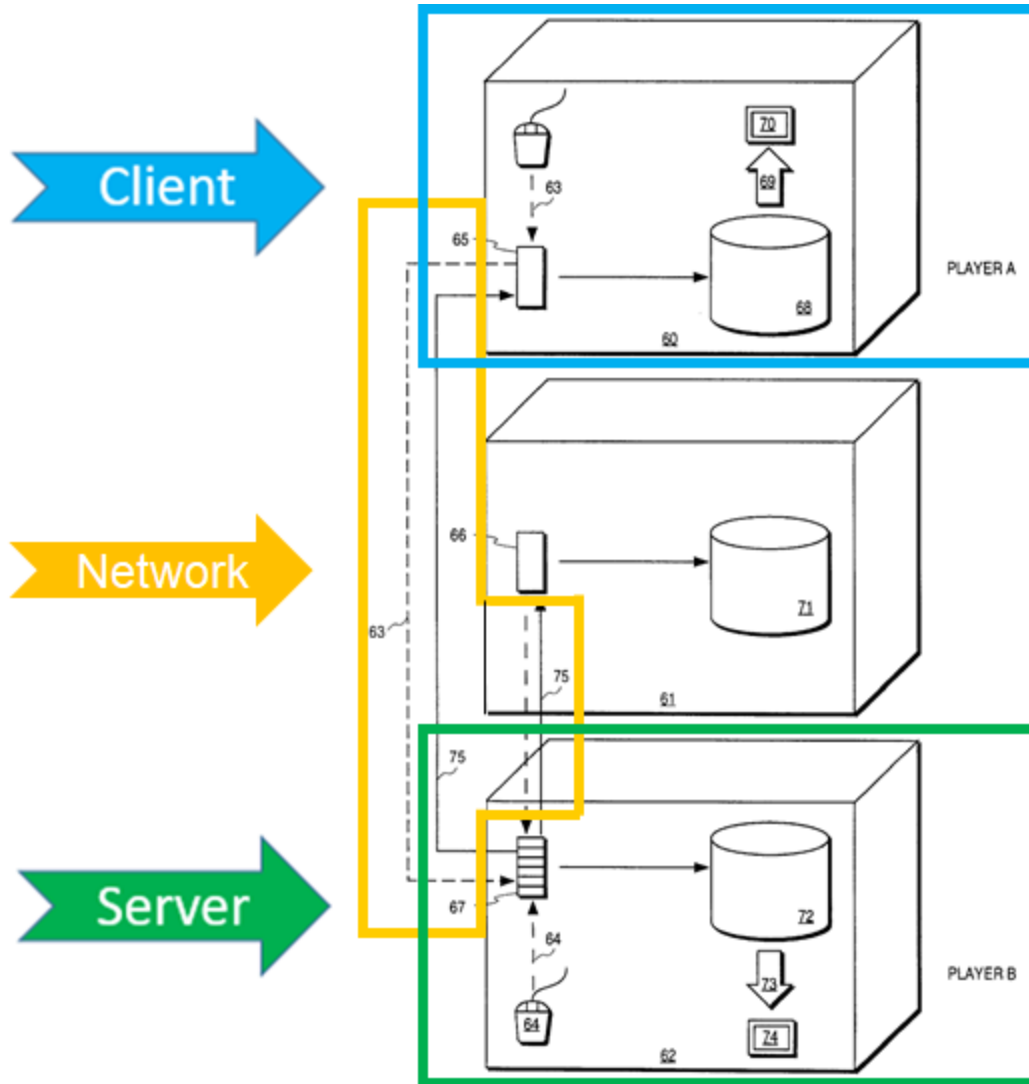


FIG. 6

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 [display 70](#) of client [computer 60](#) is updated by [rendering commands 69](#) from [slave copy 68 of the master database](#), rather than from the master database

itself, thereby reducing server interactions on the network. Ex.1003¶153; Ex.1004, Fig.6; 5:17–49.¹⁵

Further, a POSITA would understand that Roy’s view-generating logic would inherently be the same on each computer (and therefore the same whether on the server or client) for efficiency, economic, and common sense reasons. Ex.1003¶154–155. It would be particularly inherent to a POSITA because in Roy’s application, each user sees the same world and it would not make sense to use different logic to generate views of the same world for each computer, especially given Roy’s teachings that client and server computers have the same architecture. Ex.1003¶154–155; Ex.1004, Fig. 2, 3:57–59.

Third, Roy discloses “configuring the client to execute at least a subset of the [] controller logic associated with the application” on the client. Roy discloses that Player A’s client **computer system 60** executes at least a subset of the controller logic. Client **computer 60** processes **events 63** generated by user commands and **event server 65** determines whether events should be passed along to the master server, or whether they are “local events” used to update the local model stored in **database 68** and generate **rendering commands 69** causing **display**

¹⁵ While this description refers to Figure 5A, the computer 60 Player A elements are consistent with Figure 6.

70 to be refreshed. Ex.1004, 5:17–49; Ex.1003¶156–157.¹⁶ The “event server” is the component “routing events to the local database and to other event servers.” Ex.1004, 4:62–66. And every participating computer has an event server. Ex.1004, 4:37–40. Thus, the same controller logic can be executed on any computer.

Further, a POSITA would understand that the controller logic would inherently be the same on each computer (and therefore the same whether on the server or client) for efficiency, economic, and common sense reasons. Ex.1003¶158. A POSITA would understand that each computer processes “purely local events” in the same way, such as local events for setting display options. Ex.1004, 5:40–43; Ex.1003¶158.

This description meets the limitation under both proposed constructions in the Related Litigation because the controller logic program code causes the model to be changed **and** the view to be refreshed, not just one or the other. Ex.1003¶159. Accordingly, Roy discloses each limitation in (b).

4. Limitation (c)

Roy discloses one or more portions of the application (e.g., the online application) being performed at the client (e.g., **computer system 60**) without the client having to interact with the server (e.g., **computer system 62**), thereby

¹⁶ While this description refers to Figure 5A, the computer 60 Player A elements are consistent with Figure 6.

reducing interactions between client and server as required by the preamble of claims 1, 25, and 26 and limitation (c) of claim 13, as described above, the discussion of which is incorporated herein. Ex.1003¶160.

Roy discloses embodiments wherein certain events are “purely local” and processed only on the client system, thereby eliminating roundtrips to the server. Ex.1004, 5:17–49. For example, Roy’s Figure 8A flowchart requires that if the step 803 “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):”

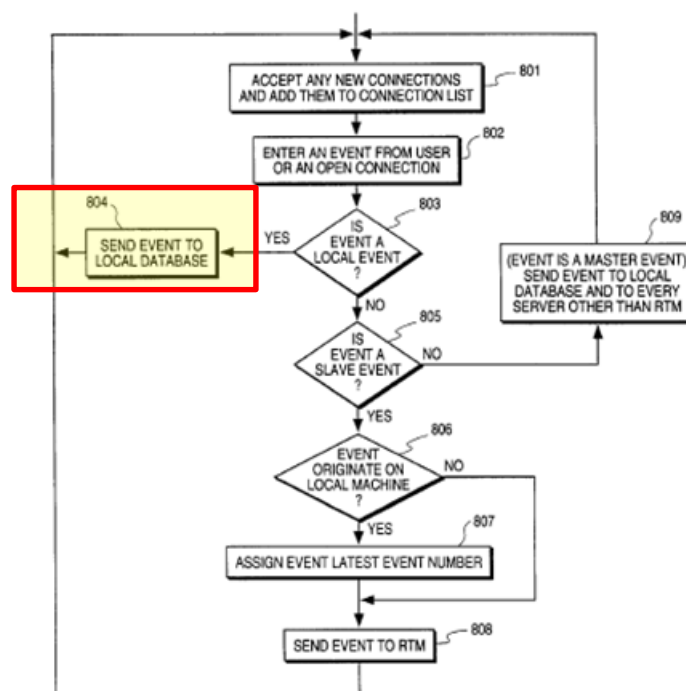


FIG. 8A

Ex.1004, Fig.8A (annotated), 6:46–67. “Local events” are processed at the client without having to interact with the server. Ex.1004, Fig.8A, 5:29–46, 6:46–67; Ex.1003¶161. Accordingly, Roy discloses each element of limitation (c). Ex.1003¶162.

5. Limitation (d)

Roy discloses that “the client and the server both locally maintain at least a portion of the model” (e.g., a portion of the application model), and execute the view-generating and controller logic associated therewith, as described above regarding claim elements (a) and (b), the discussion of which is incorporated herein. Ex.1003¶163.

In the model, view-generating logic, and controller logic for claim elements (a) and (b): (1) both the **client** and **server** “maintain at least a portion of the model” (e.g., **database 68** and **database 72**), (2) the client and server “execute the view-generating . . . logic” (e.g., **rendering commands 69 updating display 70** and **rendering commands 73 updating display 74**), and (3) the client and server “execute the . . . controller logic” (e.g., **event server 65 processing events 63** and **event server 67 processing events 64**). Ex.1003¶164. A POSITA would understand that the client-side and server-side view-generating logic and controller logic are the same, as discussed above. Accordingly, Roy discloses each element of limitation (d). Ex.1003¶164.

6. Limitation (e)

Roy discloses “a model [that] comprises application data, rules, and algorithms affecting the data” (e.g., [database 68](#) and [database 72](#)). Ex.1003¶165. Roy discloses “a view [that] comprises a screen or window representation of the subset of the model that the application chooses to display” (e.g., [display 70](#) updated by [rendering commands 69](#) and [display 74](#) updated by [rendering commands 73](#)). Ex.1003¶165. Roy discloses “a controller [that] comprises the logic and processes user requests, and causes the model to be changed and the view to be refreshed” (e.g., [event server 65](#) handling [events 63](#) and [event server 67](#) handling [events 64](#)). Ex.1003¶165. Each of these limitations was discussed above regarding the preamble and limitations (a) and (b), and that discussion is incorporated herein. Accordingly, Roy discloses limitation (e), and thus discloses every limitation of the ’719 Patent independent claims. Ex.1003¶165.

B. Ground 2: Roy Renders Obvious Claims 1, 13, 25, and 26

For the reasons stated above with respect to **Ground 1**, the independent claims of the ’719 Patent are also invalid under §103.¹⁷ Roy is analogous art to the ’719 Patent as discussed in §IV.D.1; Ex.1003¶166.

¹⁷ “[I]t is well settled that a disclosure that anticipates under §102 also renders the claim invalid under §103, for anticipation is the epitome of obviousness.” *Realtime Data, LLC v. Iancu*, 912 F.3d 1368, 1373 (Fed. Cir. 2019) (internal citations omitted).

Per agreed claim constructions in the Related Litigation, “the client and the server both locally . . . execute the view-generating and controller logic” means that the “view-generating logic” and “controller logic” executed on the client must be the same “view-generating logic” and “controller logic” executed on the server, or some subset thereof. Ex.1003¶167

A POSITA would inherently understand that Roy’s “view-generating logic” and “controller logic” are the same whether on the master/server or slave/client system, as any POSITA would seek to employ code reuse and avoid the expense and inefficiency of developing different logic and program code to perform the same task when technically feasible, as a matter of common sense and good engineering practice. Ex.1003¶¶154–155, 167; *B/E Aerospace, Inc. v. C&D Zodiac, Inc.*, 962 F.3d 1373, 1379–80 (Fed. Cir. 2020) (affirming obviousness determination because “it would have been a matter of common sense” to create a second recess in a plane design). Accordingly, to the extent Roy does not invalidate the independent claims under §102, Roy renders the claims obvious under §103.

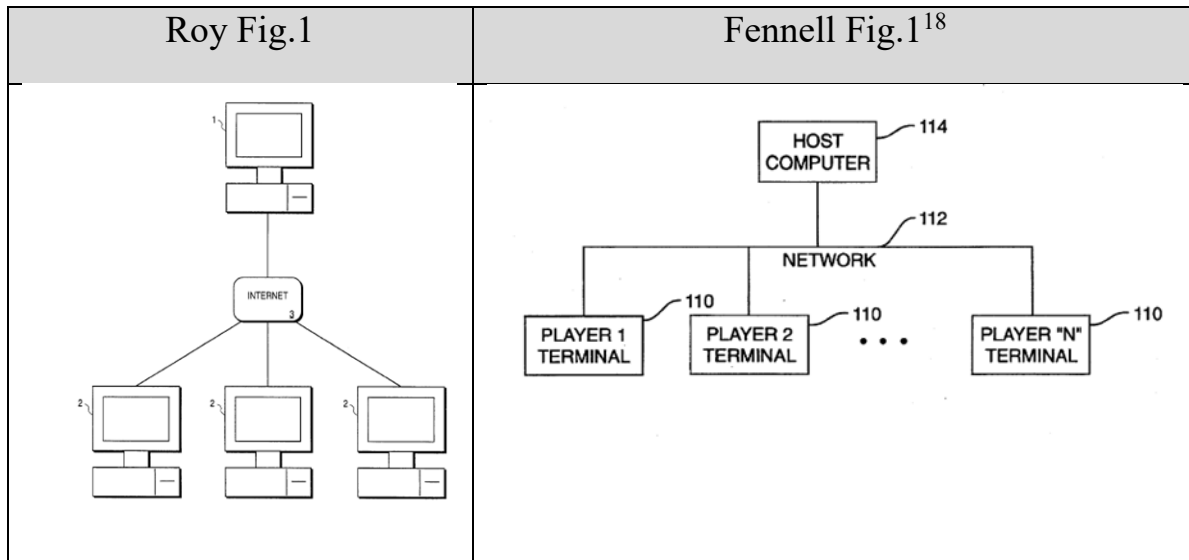
C. Ground 3: Roy, Fennell, and O'Reilly Render Obvious Dependent Claims 2, 4–12, 14, 16–24

1. Motivation to Combine

a. Roy and Fennell

A POSITA would have been motivated to modify Roy's online application based on Fennell's browser-enabled application. Ex.1003¶170. Both concern managing network latency effects in online systems and minimizing server interaction. Ex.1004, Abstract; Ex.1005, 1:39–43.

Both networked systems use an Internet connection. Ex.1003¶170–174.



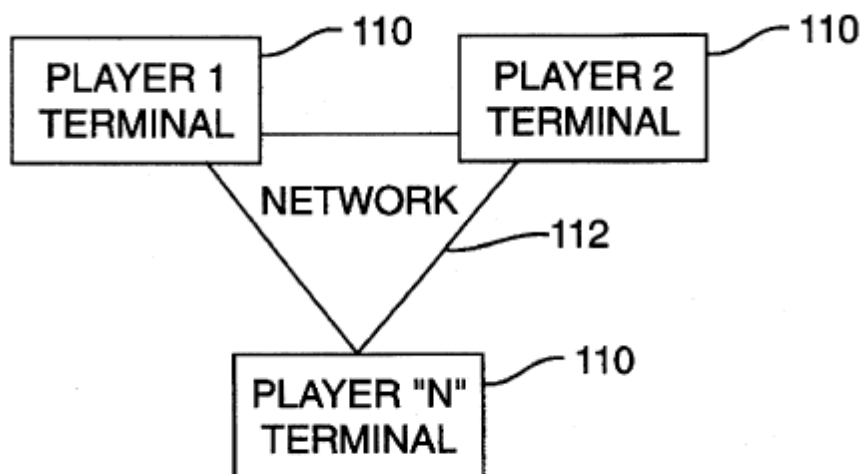
A preferred embodiment of Fennell's terminal is "a smart terminal such as a World Wide Web browser with software 'applets' (written in, e.g., Sun Microsystems' Java language) downloaded on demand." Ex.1005, 3:11–19.

¹⁸ Fennell's "Network" is "the Internet." Ex.1005, 3:23–24.

Fennell suggests that many embodiments would take the form of a browser because of the low marginal cost of distributing software. Ex.1003¶175.

A POSITA examining Roy's online application would need to determine how to distribute the application to each client computer, while also examining methods for reducing network latency effects and would have understood that Fennell's application using browser software would work in Roy's system. Ex.1003¶176.

Further, Fennell, like Roy, discloses a "dual-MVC" embodiment minimizing host-computer interaction by equipping each user with a model (game challenge), view (display) and controller (terminal equipped with an input/output device). Ex.1005, 5:38–53. In that embodiment, host-computer interaction is minimized as terminal 110 downloads an encrypted question-and-answer file (the model) and sends that model to additional participants. Ex.1003¶¶177–178. The additional participants do not interact with the host computer, but only with terminal 110, via the network 112. Ex.1003¶178. Each additional participant therefore would have a copy of the model, display (view) and controller (terminal equipped with "any suitable input/output device" Ex.1005, 3:3–4):



19

Like the '719 Patent and Roy, Fennell therefore discloses a distributed system with each terminal having a model or subset thereof, view, and controller. Ex.1003¶179.

Both Fennell and Roy address challenges associated with minimizing latency effects in multiuser environments. Roy focuses on reducing latency during multiuser database synchronization by processing events locally without client/server interaction. Ex.1003¶180. Fennell addresses the synchronization of user actions in multiuser environments by assigning ranks to users, and further, eliminating host-computer interaction in peer-to-peer embodiments. Ex.1003¶180. A POSITA would recognize the synergy between these technologies to enhance

¹⁹ Fennell at Fig. 3 (Ex.1005).

the overall performance and user experience of networked multiuser applications.
Ex.1003¶180.

Both patents are in the same technical field (networked multiuser systems) and their combination would have been predictable to a POSITA. Ex.1003¶181. Applying Roy's latency reduction methods to the synchronization and host-computer-interaction-elimination method of Fennell would yield a predictably improved user-friendlier application. Ex.1003¶181. A POSITA would naturally consider Fennell's browser-enabled application for the Roy system as Fennell explains that browser-enabled applications are a preferred method for accessing the system. Ex.1003¶181. What would result is a browser-enabled system where client/server interactions are minimized as each client in the system has a copy of the model, view-generating logic, and controller logic.

Accordingly, a POSITA seeking to distribute and develop an online application with reduced network latency effects and minimized client/server interactions would examine both Roy and Fennell and understand they could deploy the Internet-based application both references describe over computers enabled with browser software. Ex.1003¶182.

b. Roy and O'Reilly

A POSITA would have been motivated to combine Roy's latency-reducing distributed online system with O'Reilly, which explained how Java and JavaScript can interact in both web browsers and web servers. Ex.1003¶183.

Roy focuses on reducing latency in multiuser environments, a crucial requirement for real-time data synchronization. Ex.1003¶184. O'Reilly discloses that JavaScript provides an event-driven and highly interactive scripting environment, essential for responsive user interfaces in web applications. Ex.1003¶184–185. Both address challenges in creating dynamic, user-interactive systems relying on efficient data handling and synchronization. Ex.1003¶184–185.

During the time of these inventions, there was a growing trend toward building interactive web applications that required both fast data synchronization and dynamic, user-responsive interfaces. Ex.1003¶185. A POSITA would naturally consider integrating a proven backend latency-reduction mechanism (from Roy) with O'Reilly's disclosures to meet these design goals. Ex.1003¶185. By implementing Roy's database synchronization methods with JavaScript-driven client interfaces, developers could ensure that data updates were reflected instantaneously across user devices, minimizing the perception of latency in shared applications. Ex.1003¶185.

A POSITA preparing an architecture and coding for an online system would have consulted O'Reilly, a readily available handbook, as it provided information, examples, and practical solutions to problems while explaining the capabilities available to a POSITA. Ex.1003¶186. The field of endeavor is the same for the '719 Patent, Roy, and O'Reilly—ways of efficient information handling over the Internet. Ex.1003¶186.

O'Reilly teaches that developers can use JavaScript to reduce client/server interaction. Ex.1003¶187. For example, O'Reilly explains a programmer could develop a tax calculator with JavaScript, and unlike traditional methods requiring server contact to perform computations, JavaScript could conduct local computations. Ex.1006, 23–28 (§1.5). Similarly, using JavaScript eliminated round-trips to the server when pre-processing data in user forms. Ex.1006, 20 (§1.3.3); Ex.1003¶188.

Roy, likewise, teaches a system that reduces client/server interactions, as discussed above in connection with the independent claim preamble, limitation (a), and limitation (b), which is incorporated herein. Ex.1003¶189.

A POSITA would find it obvious to combine Roy's latency-reduction techniques with the interactive capabilities of JavaScript O'Reilly describes to create a low-latency, real-time, user-interactive web application. This combination addresses complementary technical challenges and aligns with known design

trends in web development, leading to predictable improvements in performance and user experience. Ex.1003¶190.

c. Roy, Fennell, and O'Reilly

As discussed, a POSITA would have been motivated to combine Roy's and Fennell's teachings. Ex.1003¶191. A POSITA would have been further motivated to combine the resultant browser-enabled multiuser Internet application with O'Reilly for at least three reasons. Ex.1003¶191.

First, as the '719 Patent admits was known in the art, "JavaScript [could] be used to implement much of the same functionality that applets provide." Ex.1001, 3:1–13; 2:22–23; Ex.1003¶192. Fennell discloses browser-based Java applets, stating a terminal could be "a smart terminal such as a World Wide Web browser with software 'applets' (written in, e.g., Sun Microsystems' Java language) downloaded on demand." Ex.1005, 3:11–19. A POSITA would be motivated to combine O'Reilly with Fennell as O'Reilly describes how to use JavaScript on browsers to interact with Java applets. Ex.1006, 283–312 (§19) (interoperability between JavaScript and Java); Ex.1003¶192.

Second, Roy and Fennell, like the '719 Patent, were concerned with minimizing network latency effects and reducing client/server interaction. Ex.1003¶193. O'Reilly describes a tax calculator employing JavaScript to reduce

server contact by locally updating logic (Ex.1006, 23–28 (§1.5)) or by pre-processing data in user forms (Ex.1006, 20 (§1.3.3)).

Third, an Internet-based application, like Roy and Fennell describe, is necessarily dynamic, and JavaScript was developed out of a need and desire to create dynamic behavior on an HTTP web page. Ex.1003¶194. A POSITA would have likely consulted O'Reilly to learn how to enable dynamic web content. Ex.1003¶194.

Accordingly, a POSITA would have had considerable motivation to combine the teachings of Roy, Fennell, and O'Reilly. Ex.1003¶195.

2. Claims 2 and 14: “a HyperText Transport Protocol”

Claims 2 and 14 claim the use of an HTTP network, which the '719 Patent Examiner noted was well known, the '719 Patent itself admits was well known, and which O'Reilly discloses. Ex.1001, 4:58–60; Ex.1003¶196–197.

First, the Examiner took “Official Notice” that “HTTP [was] well-known in the art as taught by Gish [U.S. Patent No. 5,768,510], wherein Gish [taught] the client and server communicate over a HyperText Transport Protocol network[.]” Ex.1002, 316–17; Ex.1007 (Gish); Ex.1003¶198.

Second, the '719 Patent admits that HTTP was “**well known in the art.**” Ex.1001, 4:58–60. A POSITA from the relevant time would understand that

any web-based client/server system could use HTTP for communication.

Ex.1003¶199.

Finally, the combination of Roy, with browser-based disclosures of Fennell, combined with O'Reilly, render this limitation obvious. Ex.1003¶200. Throughout the book, O'Reilly presupposes the readers' familiarity with HTTP. Ex.1003¶200. For example, O'Reilly teaches that cookies could provide the state information missing from the stateless HTTP protocol of the Web. Ex.1006, 21 (§1.3.5). O'Reilly further explains how to handle certain HTTP requests in Server-Side JavaScript. Ex.1006, 32 (§1.6.5). O'Reilly further discusses implementation of cookies as "an extension to the HTTP protocol." Ex.1006, 234 (§15.1). In short, O'Reilly presupposes familiarity with HTTP because it was well known. Ex.1003¶200.

Accordingly, claims 2 and 14 are rendered obvious. Ex.1003¶201.

3. Claims 4 and 16: "partition/ing a screen area associated with the browser software into frames"

A POSITA would have been motivated to combine Roy, Fennell, and O'Reilly to create a browser-enabled Internet-based distributed application.

§V.C.1; Ex.1003¶202–204. O’Reilly expressly discloses partitioning a screen area associated with browser software into frames.²⁰

If claim 4 refers to “configuring step” (b), O’Reilly discloses a browser window capable of partitioning a screen area associated with the browser into frames, noting that browsers can display multiple HTML pages “within a single window by using frames—a feature that allows a single browser window to be divided horizontally or vertically (or both) into individual sections that each display a separate HTML document.” Ex.1006, 184 (§11.3). A POSITA would understand that to enable this feature, a system would necessarily require a client processor that is operative to partition a screen area associated with the browser software into frames (claim 16). Ex.1003¶205–207.

If the configuring step of claim 4 refers to claim limitation (a), as opposed to (b), O’Reilly still discloses the limitation. Ex.1003¶208. O’Reilly discloses that server-side JavaScript can dynamically generate HTML to be displayed by the client, which would necessarily be enabled to display that information in frames as noted above. Ex.1006, 32 (§1.6.5) (“server-side JavaScript is embedded directly

²⁰ This is true whether a frame is construed as a “section of a webpage, each constituting a distinct HTML document” or “plain and ordinary meaning.” Ex.1003¶205n.164.

within HTML pages;” “An obvious capability of server-side JavaScript is to dynamically generate HTML to be displayed by the client.”).

Accordingly, a POSITA seeking to distribute and develop an Internet-based browser-enabled application as Roy and Fennell disclose would understand that partitioning a screen area associated with the browser software into frames was known. Ex.1003¶209. Thus Roy, in combination with Fennell and O’Reilly renders claims 4 and 16 of the ’719 Patent obvious.

- 4. Claims 5 and 17: “wherein the at least a subset of the model, the view-generating and the controller logic associated with the application are associated with at least one frame and one or more views for display in accordance with the application are associated with at least another frame”**

A POSITA would have been motivated to combine Roy, Fennell, and O’Reilly to create a browser-enabled Internet-based distributed application. §V.C.1; Ex.1003¶¶210–212. As discussed above in connection with the independent claim preamble and limitations (a) and (b), Roy discloses a “dual-MVC” approach that places a copy of the model, view-generating logic and the controller logic on a client. §V.A.1–3; Ex.1003¶213. A POSITA would find it obvious, based on the teachings of O’Reilly, to associate the model, view-generating logic, and controller logic with a frame that is separate from the viewed frame. Ex.1003¶213.

First, O'Reilly discloses that JavaScript can be used to associate a subset of a model associated with an application in a frame. O'Reilly provides an example of using JavaScript to associate tax bracket information with a frame. Ex.1006, 23–28, 208) (§1.5; §12.4); Ex.1003¶214.

Second, O'Reilly discloses that JavaScript can be used to associate view-generating logic with a frame, and expressly states that “one commonly used feature of JavaScript is the ability to dynamically generate HTML and JavaScript content for display in other browser windows and frames.” Ex.1006, 164 (§10.1.2, FNf). In this example, the JavaScript code creating a dynamic HTML document is view-generating logic, because the HTML document that is displayed is a result of the JavaScript code specifying the content and look of the HTML document. Ex.1003¶215.

Third, O'Reilly discloses that JavaScript can be used to associate controller logic with a frame. Ex.1003¶216. Specifically, O'Reilly teaches frame programming techniques that deploy multiple frames, at least one of which can run a function and at least another of which can dynamically display the results of that function. Ex.1006, 208 (§12.4):

Once you know how to refer to any frame from any other frame, you can start writing JavaScript programs that work in complex framed documents. . . . For example, if an event handler in frame A invokes a function that is defined in frame B, the code in that function is running in frame A, not frame B—and if the code wants to refer to frame B, it

can't just use the implicit window reference, as it could if it were actually running in frame B.

In this example, the “event handler” in frame A can be considered controller logic, as O'Reilly defines “event handlers” as “pieces of JavaScript code that are executed when a particular event occurs, such as when the user clicks on a button.” Ex.1006, 15 (§1.1); Ex.1003¶217.

Fourth, O'Reilly teaches that the frame associated with the model, view-generating logic, and controller logic should preferably be associated with at least one frame that is different from the frame associated with a view. Ex.1003¶218. O'Reilly teaches that it is actually *harder* for JavaScript to create a view in the same frame associated with the logic to create the view, because doing so would overwrite the JavaScript code itself. Instead, O'Reilly teaches that it is beneficial to separate the display into its own frame for dynamic updates (Ex.1006, 208 (§12.4)), the same concept purported to be invented in claims 5 and 17:

As we saw in the error handler example, JavaScript code in one window (or frame) can dynamically create an HTML document in another window (or frame). It is a lot harder for JavaScript code to dynamically create a new HTML document in its own window or frame, because doing this generally overwrites the JavaScript code itself!

O'Reilly discloses an example of generating dynamic HTML content wherein the frames themselves are dynamically created (Ex.1006, 209 (§12.4)):

We'd described how you can use JavaScript running in one frame to dynamically create HTML content for another frame. But in this

discussion we have always created the frames themselves with a frameset specified in a static HTML file. Since frames are specified in HTML, there is no reason we cannot create them dynamically as well.

And O'Reilly specifies that the "write()" method can dynamically generate an HTML document that is displayed as intended, noting that it would be preferable to do so from one frame into another frame (Ex.1006, 224 (§14.1.2)):

Without a doubt, the most important feature of the Document object (and perhaps of client-side JavaScript in general) is the write() method, which allows us to dynamically generate web page content from our JavaScript programs. . . . Although you can't usefully write to the current document from an event handler, there is no reason you can't write to a document in another window or frame, and doing so can be a very useful technique for multiwindow or multiframe web sites.

O'Reilly therefore teaches that it is beneficial to have a "view," or a dynamically created HTML document, associated with a frame that is different from the frame that creates the view. Ex.1003¶¶219–221. Otherwise, O'Reilly teaches, application code would be overwritten. Ex.1003¶221.

Frames were already a well-known tool for isolating functionality in browser-based systems, as O'Reilly teaches. Ex.1003¶222. Associating different application components (logic vs. views) with separate frames would have been a predictable implementation for a modular, browser-based client-server system the combination of Roy and Fennell disclose. Ex.1003¶222.

Accordingly, a POSITA would understand that to deploy the Internet-based browser-enabled application Roy and Fennell teach, it would be beneficial to separate the subset of the model, view-generating and controller logic associated with the application into one frame, and the resultant views for display into another frame, as O'Reilly teaches. Ex.1003¶223. Thus Roy in combination with Fennell and O'Reilly renders obvious claims 5 and 17 of the '719 Patent. Ex.1003¶224.

5. Claims 6 and 18: “wherein the at least one view frame is a visible frame”

A POSITA would have been motivated to combine Roy, Fennell, and O'Reilly to create a browser-enabled Internet-based distributed application. §V.C.1; Ex.1003¶225–227.

A POSITA would understand that in order to have a workable Internet-based browser-enabled application, the view of the application would necessarily need to be visible to the user, which O'Reilly discloses. Ex.1003¶228. For example, O'Reilly discloses a browser window capable of **displaying** HTML documents. Ex.1006, 184 (§11.3).

O'Reilly teaches that it is beneficial to separate the display into its own frame for dynamic updates, and distinguished visible frames from invisible ones. Ex.1006, 207–11 (§12.4); Ex.1003¶229–230. Accordingly, a POSITA would understand that to deploy the Internet-based browser-enabled application Roy and

Fennell teach, the view must be visible as O'Reilly teaches, rendering obvious claims 6 and 18. Ex.1003¶231.

6. Claims 7 and 19: “wherein the at least one frame associated with the at least a subset of the model, the view-generating logic and the controller logic is not a visible frame”

A POSITA would have been motivated to combine Roy, Fennell, and O'Reilly to create a browser-enabled Internet-based distributed application. §V.C.1. Specifically, O'Reilly teaches frame programming techniques that deploy multiple frames, at least one of which can run a function and at least another of which can dynamically display the results of that function. Ex.1006, 207–211 (§12.4); Ex.1003¶¶232–234.

O'Reilly further teaches that it is actually *harder* for JavaScript to create a view in the same frame associated with the logic to create the view, and thus recommended that a programmer use an *invisible* frame, which O'Reilly discloses how to do. Ex.1006, 208–09 (§12.4). In O'Reilly's example, the invisible frame is not visible because it has a height of zero. Ex.1003¶236. The example's invisible frame refers to a separate JavaScript program (“program.html”) that dynamically updates the visible frames. Ex.1003¶238.

As discussed above in connection with the independent claim preamble and limitations (a) and (b), Roy discloses a “dual-MVC” approach that associates a copy of the model, view-generating logic and controller logic with a client.

§V.A.1–3; Ex.1003¶239. A POSITA would understand that to avoid overwriting the subset of the model, view-generating logic and controller logic, it would be beneficial to associate those parts of the program with an invisible frame.

Ex.1003¶239. Otherwise, the frame could be overwritten and lose the ability to run the application Roy discloses. Ex.1003¶239.

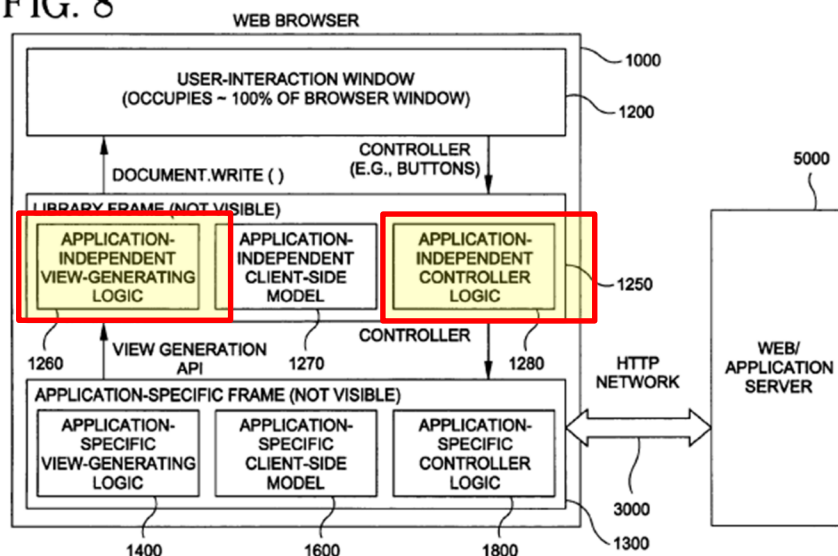
Thus, Roy in combination with Fennell and O'Reilly, renders obvious claims 7 and 19. Ex.1003¶240–241.

7. Claims 8 and 20: “form/ing at least one frame with which application-independent view-generating logic and controller logic is associated”

A POSITA would have been motivated to combine Roy, Fennell, and O'Reilly to create a browser-enabled Internet-based distributed application. §V.C.1; Ex.1003¶242–245. Roy discloses application-independent view generating logic and controller logic, and O'Reilly discloses associating that logic with a frame.

The '719 Patent discloses “application-independent” view-generating logic (1260) and controller logic (1280). Ex.1001, Fig.8 (highlighting added).

FIG. 8



What the '719 Patent characterizes as examples of “application-independent controller logic” (“generic user-interface event handling”) and “application-independent view-generating logic” (“logic to generate a button on the screen”) were well-known, widely practiced, and routinely taught in undergraduate classrooms decades before the '719 Patent was filed. Ex.1001, 7:51–63; Ex.1003¶¶243, 246.

Roy discloses “application-independent view-generating logic and controller logic.” Roy discloses “purely local” display options that relate to, for example, “events which set display options or other functions that are only relevant to” a particular computer system. Ex.1004, 5:17–49. These include application-agnostic view-generating logic, and could relate to, for example, the display’s background color or text color, which would be processed by purely local controller logic.

Accordingly, a POSITA would understand that Roy discloses application-independent view-generating and controller logic. Ex.1003¶¶248–249.

O'Reilly further discloses forming “at least one frame with which application-independent view-generating logic and controller logic is associated.” Ex.1003¶250.

First, O'Reilly discloses a library of view-generating logic attributes a developer could call on to create an HTML document, such as specifying the colors it is displayed in. Ex.1006, 222–24 (§14.1); Ex.1003¶251.

O'Reilly further explains view-related attributes, including how to specify background color and text for a document. Ex.1006, 222–23 (§14.1.1); Ex.1003¶252.

And O'Reilly specifies that the “write()” method can dynamically generate an HTML document that is displayed as intended, noting it would be preferable to do so from one frame into another frame. Ex.1006, 224–25 (§14.1.2); Ex.1003¶253.

Accordingly, a POSITA would understand that application-independent view-generating logic would preferably be associated with a frame other than the document frame, and would deploy this strategy in an Internet-based browser enabled application. Ex.1003¶254.

Second, O'Reilly discloses "application-independent . . . controller logic." O'Reilly describes a number of "event handlers," which the '719 Patent provides is an example of "application-independent . . . controller logic." Ex.1001, 7:61–63.; Ex.1003¶255.

O'Reilly provides that event handlers are an "important feature" of JavaScript, and even provides a table of Supported Event Handlers. Ex.1006, 21; 166–69 (§1.3.4; §10.3.2 ("Event Handlers in JavaScript"; Table 10-1 ("JavaScript Event Handlers")); Ex.1003¶256.

And O'Reilly specifies that the "write()" method can be called using event handlers, even noting it would be preferable to do so from one frame into another frame. Ex.1006, 224 (§14.1.2); Ex.1003¶257.

A POSITA would understand that "application-independent . . . controller logic," like handling mouse clicks, would preferably be associated with a frame separate and apart from the document frame itself, per the teachings of O'Reilly, and would deploy this strategy in an Internet-based browser enabled application. Ex.1003¶258.

Roy, Fennell, and O'Reilly collectively emphasize modularity, enabling reuse of components across applications. Associating application-independent logic with a frame aligns with this shared objective. Ex.1003¶259.

In particular, frames, as O'Reilly teaches, were a known tool for encapsulating and isolating reusable logic. Ex.1003¶260. A POSITA would recognize that associating such logic with a frame simplifies application development and maintenance, which aligns with the goals of Roy and Fennell. Ex.1003¶260.

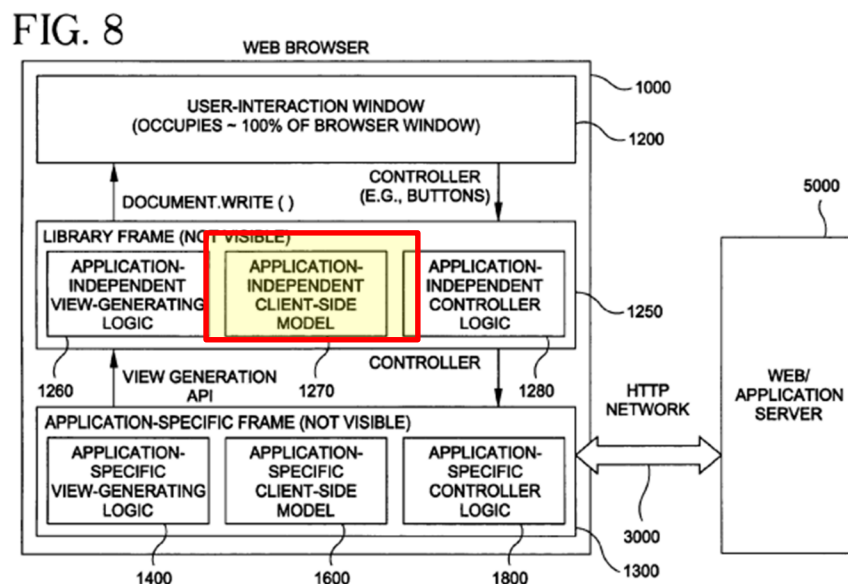
Roy and Fennell provide methods for decentralizing logic and improving application efficiency. Ex.1003¶261. O'Reilly complements these teachings by providing a mechanism (frames) to encapsulate such logic, making it reusable and application-independent. Ex.1003¶261. Developers sought to reduce redundancy and improve scalability in client-server systems. Ex.1003¶261. Associating reusable, application-independent logic provides an obvious solution for achieving these objectives, enhancing both development efficiency and application performance. Ex.1003¶261.

Accordingly, a POSITA would find it obvious to have at least one frame with which application-independent view-generating logic and controller logic is associated, as O'Reilly teaches, in the Internet-based application Roy and Fennell teach, rendering obvious claims 8 and 20. Ex.1003¶263.

8. Claims 9 and 21: “wherein the at least one application-independent view-generating logic and controller logic frame further has an application-independent model associated therewith”

A POSITA would have been motivated to combine Roy, Fennell, and O'Reilly to create a browser-enabled Internet-based distributed application. §V.C.1; Ex.1003¶¶264–267. Roy and O'Reilly disclose an application-independent model, and O'Reilly discloses associating that model with a frame.

The '719 Patent's example of an “application-independent client-side model” is “caching and data-movement operations.” Ex.1001, Fig.8 (highlighted), 7:51–63; Ex.1003¶¶268–269.



But caching and data-movement operations were well-known and widely practiced decades before the '719 Patent was filed. Ex.1003¶270. Caching was

developed and disclosed in 1965 (Ex.1012 (Wilkes) (referred to as “slave memory”)), and IBM even sought an early 1970’s patent directed to caching and associated data movement. Ex.1013 (Barner). Ex.1003¶270. In addition, caching and associated data-movement operations have been part of both undergraduate and graduate computer science curricula since at least 1985. Ex.1003¶270. Roy and O’Reilly also disclose these concepts.

First, Roy discloses an “application-independent model.” Roy discloses each machine other than the machine in which the master event server is located stores slave copies of the master database. Ex.1001, 4:44–54. The slave copies of the master database are cached portions of the master database. Ex.1003¶¶271–272. The slave computer system (client) performs caching operations when it updates the slave database. Ex.1003¶272. Similarly, the server performs cache-related “data-movement operations” when it sends events to slave computers across the network that cause the slave database (cache) to be updated. Ex.1003¶272. These caching and data-movement operations do not depend upon the specifics of the application performing them and are thus application-independent. Ex.1003¶272. Given the desire for browser-enabled applications as Fennell discloses, a POSITA would further be motivated to combine the teachings of Roy with O’Reilly and Fennell to associate the application-independent model with a frame. Ex.1003¶272.

Second, O'Reilly discloses an "application-independent model." O'Reilly provides significant guidance regarding employing caching and data-movement operations in JavaScript. Ex.1003¶273. For example, O'Reilly teaches that allowing JavaScript files to be cached by the browser allows them to load quicker. Ex.1006, 166 (§10.2); Ex.1003¶274.

O'Reilly also teaches a method of caching images for easy replacement into dynamic HTML pages. Ex.1006, 242 (§16.1); Ex.1003¶275.

And O'Reilly teaches that caching JavaScript improves performance, explaining that once JavaScript code is cached locally, loading and webpage execution delays effectively disappear. Ex.1006, 174 (§10.6.1); Ex.1003¶276. O'Reilly also discloses techniques for performing application independent data-movement operations using JavaScript, including saving state with cookies and sending cookies to a server (Ex.1006, 234–41 (§15)), loading the results of form submissions into a frame (Ex.1006, 251–66 (§17)), caching JavaScript code (Ex.1006, 166 (§10.2)), and image caching (Ex.1006, 242 (§16.1)). Ex.1003¶277. This provides further motivation for a POSITA to deploy O'Reilly's teachings in Roy's and Fennell's browser-enabled system. Ex.1003¶276, 278.

O'Reilly further teaches that it is preferable to associate logic that would interact with documents with a frame separate from the document frame to avoid the risk of overwriting the logic. Ex.1006, 207–09 (§12.4). Ex.1003¶279.

A POSITA would understand that an application-independent model, like a caching function, would preferably be associated with a frame separate and apart from the document frame itself, per the teachings of O'Reilly, and would deploy this strategy in an Internet-based browser enabled application. Ex.1003¶280. Accordingly, a POSITA would find it obvious that the at least one application-independent view-generating logic and controller logic frame further has an application-independent model associated therewith, as O'Reilly teaches, in the Internet-based browser-enabled application Roy and Fennell teach, rendering obvious claims 9 and 21. Ex.1003¶¶281–282.

9. Claims 10 and 22: “wherein the at least one application-independent view-generating logic and controller logic frame serves as an application programming interface for developing views to be displayed in accordance with the application”

A POSITA would have been motivated to combine Roy, Fennell, and O'Reilly to create a browser-enabled Internet-based distributed application. §V.C.1; Ex.1003¶¶283–285. Roy and O'Reilly both disclose the use of APIs, and a POSITA would find obvious that an application-independent view-generating logic and controller logic frame serves as API for developing views to be displayed in accordance with the application. Ex.1003¶286.

The '719 Patent describes an API being deployed in JavaScript, stating that the “library frame 1250 provides an [API] and implementation for generating the

view in the visible frame. . . . In a preferred embodiment, the application programmer codes the application-specific frame model and controller logic using JavaScript, and generates the view via calls to the library API.” Ex.1001, 7:51–67, 8:34–38; Ex.1003¶287.

Roy discloses the use of APIs to connect the application software to operating system software. Ex.1004, 3:48–56; Ex.1003¶288.

O’Reilly likewise discloses APIs to connect various software functionalities, for example, explaining that Java can interact with JavaScript “through the API provided by the JSObject class[,]” Ex.1006, 307 (§19.6.3), and disclosing APIs that connect JavaScript to Java applets, including LiveConnect, LiveAudio, LiveVideo, and Live3D. Ex.1006, 623–26 (Appendix H). Ex.1003¶289.

A POSITA would have found it obvious to use the application-independent view-generating logic and controller logic frame as an API for developing views to be displayed in accordance with the application, because the API would call the appropriate methods to generate the views needed to run the application.

Ex.1003¶290. As Roy discloses, its software APIs enabled interfacing with higher-level software, e.g., a runtime system. Ex.1003¶290. This would be required to provide a view of the model so that a user could actually interact with the application. Ex.1003¶290. Combined with the teachings of O’Reilly—that JavaScript APIs could connect to various software elements—it would be obvious

to deploy the application-independent view-generating logic and controller logic frame as an API for developing views to be displayed in accordance with the application. Ex.1003¶290.

O'Reilly renders obvious frames serving as an application-independent view-generating and controller logic container, and describes using frames to modularize and isolate reusable application logic. Ex.1003¶291. By combining this teaching with the synchronization and client-local execution models of Roy and Fennell, a POSITA would predictably arrive at the claimed invention.

Ex.1003¶291. Frames inherently allow independent operation and communication with other parts of the application, making them natural candidates to contain APIs for view generation. Ex.1003¶291.

Combining the modular synchronization and local execution methods in Roy and Fennell with the teaching of frames as modular, reusable containers in O'Reilly renders obvious these claims. Ex.1003¶292. A POSITA would be motivated to configure a frame to act as an application-independent API for view generation and controller logic, as this would predictably reduce server interactions, improve modularity, and streamline application development and maintenance in client-server systems, rendering obvious claims 10 and 22. Ex.1003¶¶292–293.

10. Claims 11 and 23: “wherein the views are implemented in accordance with the HyperText Markup Language and the application programming interface is implemented in accordance with the JavaScript language”

A POSITA would have been motivated to combine Roy, Fennell, and O’Reilly to create a browser-enabled Internet-based distributed application. §V.C.1; Ex.1003¶¶294–297. O’Reilly discloses implementing views in HTML and the API in JavaScript.

First, O’Reilly discloses a number of view-related features of HTML documents, and a POSITA would thus understand that the view would be implemented in HTML. For example, O’Reilly discloses using the “write() method” to dynamically generate web page content from JavaScript programs to “output HTML into the document that is currently being parsed.” Ex.1006, 224 (§14.1.2); Ex.1003¶298. O’Reilly explains the view-related attributes as specifying background and text color. Ex.1006, 222–23 (§14.1.1); Ex.1003¶299.

And O’Reilly specifies that the “write()” method can dynamically generate an HTML document that is displayed as intended, even noting it would be preferable to do so from one frame into another frame. Ex.1006, 224 (§14.1.2); Ex.1003¶300.

Second, though claimed as novel in claims 11 and 23, a POSITA would already know to implement the claimed API in JavaScript. For example, O’Reilly

discloses the use of JavaScript APIs to connect Java “through the API provided by the JScript class.” Ex.1006, 307 (§19.6.3). O’Reilly also discloses a number of APIs that connect JavaScript to Java applets, including LiveConnect, LiveAudio, LiveVideo, and Live3D. Ex.1006, 623–26 (Appendix H); Ex.1003¶301.

Further, a POSITA would understand that JavaScript and HTML work together. For example, O’Reilly explains that both JavaScript and HTML would be used to create framed layouts, stating that JavaScript’s “ability to dynamically generate HTML allows you to programmatically write the HTML tags that will create any desired frame layout.” Ex.1006, 19 (§1.3.2); Ex.1003¶¶302.

Accordingly, a POSITA would find it obvious that the views of the ’719 Patent would be implemented in accordance with the HyperTextMarkup Language and the application programming interface would be implemented in JavaScript, as O’Reilly teaches, in the Internet-based browser-enabled application Roy and Fennell teach, rendering obvious claims 11 and 23. Ex.1003¶¶303–304.

11. Claims 12 and 24: “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”

A POSITA would have been motivated to combine Roy, Fennell, and O’Reilly to create a browser-enabled Internet-based distributed application. §V.C.1; Ex.1003¶¶305, 308–309. Roy, O’Reilly, and Fennell all disclose the

concept of downloading “on demand,” rendering claims 12 and 24 obvious.

Ex.1003¶310.

In addition, a POSITA would have extensive knowledge and experience downloading a model, as well as view-generating and controller logic associated with that model, since commercial web browsers included that capability long before the ’719 Patent was filed. Ex.1003¶¶306–307. The on-demand downloading of web pages—employing HTML and JavaScript—from a web server to a web browser performed exactly that function. And such web pages were already widely supported at least by 1996. Ex.1003¶307.

Roy discloses that server data (including the model) can be migrated to any client computer on demand if requested. Ex.1003¶311. When a new user joins a session, a slave copy of the database is transferred to the new user’s computer (i.e., at least a subset of the model would be downloaded from the server to the client upon demand). Ex.1004, 4:51–55; Ex.1003¶¶312–314. A POSITA would understand that Roy’s goal of “reducing bandwidth consumption on the network,” would be accomplished by only downloading data “on demand.” Ex.1004, 5:17–49; Ex.1003¶315.

O’Reilly suggests that downloading files takes time and can cause delays, and accordingly, a POSITA would know to only download the data if it was actually needed (or in demand). Ex.1006, 174 (§10.6.1); Ex.1003¶316.

Fennell expressly teaches that online applications could be deployed over a browser with “on demand” software. Specifically, Fennell teaches that in its networked system, a “terminal” was “a browser with software ‘applets’ (written in, e.g., Sun Microsystems’ Java language) downloaded on demand[.]” Ex.1005, 3:11–15 (emphasis added); Ex.1003¶317.

Accordingly, a POSITA would find it obvious that at least a subset of the model, view-generating and controller logic associated with the application are downloaded from the server to the client upon demand, rendering claims 12 and 24 obvious. Ex.1003¶¶318–319.

D. Ground 4: Roy and Fennell Render Obvious Dependent Claims 3 and 15 Requiring “Browser Software”

The ’719 Patent is agnostic to the type of “browser software” that claims 3 and 15 require. Ex.1001, 4:51–55. Accordingly, any browser software disclosed would render obvious these claims. Ex.1003¶¶320–321.

As discussed above, Fennell discloses that an Internet-based application could be deployed over a browser with software applets. Ex.1005, 3:11–15. Ex.1003¶322. A POSITA would have been motivated to combine Roy and Fennell, as discussed above. Accordingly, the combination of Roy and Fennell renders obvious that the application could be operated such that the client performs the one

or more portions of the application in accordance with browser software running thereon, rendering obvious claims 3 and 15. Ex.1003¶322.

VI. SECONDARY CONSIDERATIONS

No known secondary considerations of nonobviousness having the requisite nexus exist to negate the '719 Patent's invalidity. Ex.1003¶324.

If IBM tries to rely on purported prior art solutions as demonstrating a long-felt but unmet need, Ex.1001, 1:9–2:67, it could not successfully do so. For example, the '719 Patent claims its solution is superior to a number of prior art systems including “dumb terminals,” “fat-client applications,” “applets,” “‘thin-client’ application architecture,” “DHTML,” “HTML (HyperText Markup Language) which includes JavaScript and Cascading Style Sheets.” Ex.1003¶325.

However, the alleged improvements over these technologies claimed in the '719 Patent already existed at the time the patent was filed. Thus, there was no long felt but unmet need. *See Robert Bosch, LLC v. Iancu*, 778 Fed. App'x 871, 881 (Fed. Cir. 2019) (no long-felt need solved by the patent when “the prior art already provided wiper blades with the claimed components”); *Merck & Cie v. Gnosis S.p.A.*, 808 F.3d 829, 838 (Fed. Cir. 2015) (patent did not resolve a long-felt need where prior art disclosed subject matter claimed).

Further, the '719 Patent's purported benefit of not having “application installation or persistence on the client” is not actually claimed. Ex.1001, claims 1–

26 (neither “application installation” nor “persistence on the client” claimed). And as discussed above, Roy, Fennell, and O’Reilly together disclose all the subject matter that *is* claimed, including a “dual-MVC” system that reduces server interaction, HTTP (e.g., Claims 2 and 14), browser software (Claims 3 and 15), frames (e.g., Claims 4–6, 8–10, 16–18, 20–22), invisible frames (e.g., Claims 7 and 19), HTML and JavaScript (e.g., Claims 11 and 23), API controls (e.g., Claims 10 and 22), downloading on demand (e.g., Claims 12 and 24), and application independent features including caching, view-generating logic, and controller logic (e.g., Claims 8–10, 20–22). §§V.A; C–D; Ex.1003¶326.

In contrast to this evidence that others successfully implemented the ’719 Patent claimed subject matter, Petitioner is unaware of evidence of failure of others practicing these claims, especially as multiple prior art references disclose the successful implementation of the subject matter claimed. Ex.1003¶327; §§IV.D.1–3; V.A–D.

No known evidence exists of commercial success with the required nexus that would affect the claims’ invalidity. Ex.1003¶¶328–329. IBM asserted the ’719 Patent against third-party gaming company Zynga Inc. With respect to the ’719 Patent, a jury entered an award for infringement of only claim 1.²¹ But as discussed

²¹ *International Business Machines Corp. v. Zynga Inc.*, Case No. 22-cv-590-GBW, D.I. 529 (D. Del.) (Ex.1017).

above, Roy anticipates claim 1 of the '719 Patent, §V.A, and thus the Zynga '719 Patent jury verdict is not relevant to the secondary considerations analysis. In addition, “the Board has repeatedly found infringement verdicts to have minimal value in establishing objective indicia of nonobviousness.” *Geotab USA, Inc. v. Omega Patents, LLC*, 2024 Pat. App. LEXIS 2628, *56, IPR2023-00504, Paper 45 (PTAB July 18, 2024) (citations omitted).

Moreover, neither O'Reilly nor Fennell, which provide strong evidence of obviousness, were raised during the *Zynga* litigation or trial. Thus, secondary considerations cannot overcome the overwhelming case of obviousness presented here. *Bosch*, 778 Fed. App'x at 881 (“sparse evidence of secondary considerations does not outweigh the case for obviousness based on the prior art”); *Merck*, 808 F.3d at 838 (“If commercial success is due to an element in the prior art, no nexus exists.”) (citation omitted).

Further, as to licensing, during the *Zynga* trial, there was unrebutted testimony that IBM had not signed a single-patent license to the '719 Patent alone, but only included the '719 Patent in portfolio licenses. Ex.1003¶328; Ex.1018 (Trial Tr. at 126:22–127:1). Portfolio licenses cannot demonstrate a nexus to the claims of the '719 Patent. *See, e.g., Merck & Cie v. Gnosis S.p.A.*, 808 F.3d 829, 838 (Fed. Cir. 2015) (affirming Board's determination that evidence should not be given “much weight” when the patent was one of many licensed); *Geotab*, 2024

Pat. App. LEXIS 2628, *53 (Board afforded little weight to licensing evidence when “dozens of patents and pending applications” were included in license). There is no known evidence that Zynga’s commercial activity has the required nexus to the claims of the ’719 Patent to negate their invalidity.

VII. DISCRETIONARY DENIAL UNDER *FINTIV* IS INAPPROPRIATE

No trial date has been set in the Related Litigation, nor has a *Markman* order issued. Discretionary denial under *Apple Inc. v. Fintiv, Inc.*, IPR2020-00019 (PTAB Mar. 20, 2020) is therefore inappropriate. And Petitioner has voluntarily presented a *Sotera* stipulation (Ex.1015) demonstrating that discretionary denial would be further inappropriate. For at least these reasons, and considering the compelling merits of the above-detailed grounds, the Board should not discretionarily deny instituting this petition.

VIII. DISCRETIONARY DENIAL UNDER §325(d) IS INAPPROPRIATE

None of the prior art that forms the basis of the Petition’s Grounds was before the USPTO during prosecution, and thus denial on this basis would not be appropriate. 35 U.S.C. §325(d).

IX. CONCLUSION

Because claims 1–26 of the ’719 Patent are invalid in view of the grounds specified in this Petition, Carvana respectfully requests institution of IPR and cancellation of the Challenged Claims.

Date: January 31, 2025

Respectfully submitted,

BALLARD SPAHR LLP
Customer Number 23859
(678) 420-9300 Phone
(678) 420-9301 Fax

By: */s/ Charley F. Brown*
Charley F. Brown
Reg. No. 52,658
browncf@ballardspahr.com

Ballard Spahr LLP
999 Peachtree Street, NE, Suite 1600
Atlanta, GA 30309

CERTIFICATE OF COMPLIANCE

Pursuant to 37 C.F.R. § 42.24(d), the undersigned certifies that the foregoing
PETITION FOR INTER PARTES REVIEW OF U.S. PATENT NO. 7,702,719,
exclusive of the parts exempted as provided in 37 C.F.R. § 42.24(a), contains
13,957 words and therefore complies with the type-volume limitations of 37 C.F.R.
§ 42.24(a).

Date: January 31, 2025

By: */s/ Charley F. Brown*

Charley F. Brown
Reg. No. 52,658
browncf@ballardspahr.com

Ballard Spahr LLP
999 Peachtree Street, NE, Suite 1600
Atlanta, GA 30309

CERTIFICATE OF SERVICE

I hereby certify that that on January 31, 2025, I caused true and correct copy of the foregoing **PETITION FOR INTER PARTES REVIEW OF U.S. PATENT NO. 7,702,719** to be served on the following counsel for International Business Machines Corporation:

Tamir Packin
Karim Z. Oussayef
William N. Yau
Michael Hilyard
Lindsey Miller
Jordan Malz
Asim Zaidi
DESMARAIS LLP
230 Park Avenue
New York, NY 10169
Tel: 212-351-3400
Fax: 212-351-3401
tpackin@desmaraisllp.com
koussayef@desmaraisllp.com
wyau@desmaraisllp.com
mhilyard@desmaraisllp.com
lmiller@desmaraisllp.com
jmalz@desmaraisllp.com
azaidi@desmaraisllp.com

Date: January 31, 2025

By: /s/ Charley F. Brown