UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

DISNEY ENTERTAINMENT & SPORTS LLC,
Petitioner

v.

ADEIA GUIDES INC.,
Patent Owner.

Case No. IPR2026-00053
U.S. Patent No. 9,235,428

DECLARATION OF DR. BRIAN W. ANTHONY IN SUPPORT OF PETITION FOR *INTER PARTES* REVIEW OF U.S. PATENT NO. 9,235,428

TABLE OF CONTENTS

Page

TABLE OF CONTENTS
(continued)

Page

TABLE OF CONTENTS
(continued)

Page

TABLE OF CONTENTS
(continued)

Page

TABLE OF CONTENTS
(continued)

Page

TABLE OF CONTENTS
(continued)

Page

I. INTRODUCTION

1. I have been retained by Disney Entertainment & Sports LLC ("Petitioner"), as an independent expert in this proceeding before the Patent Trial and Appeal Board ("PTAB" or "Board"). I understand that Petitioner is requesting that the Board institute an inter partes review ("IPR") proceeding of U.S. Patent No. 9,235,428 ("the '428 Patent") (Ex 1001), currently assigned to Adeia Guides Inc. ("Patent Owner" or "PO").

2. I am not and have never been an employee of Petitioner. I am being compensated at my usual and customary rate of $500 per hour. No part of my compensation depends on the outcome of this proceeding, and I have no other interest in this proceeding.

3. I have been asked to provide my independent analysis of the '428 Patent in light of the prior art publications cited below. I have also been asked to consider the state of the art and prior art available as of February 1, 1999. Based on the prior art discussed in this declaration, it is my opinion that Claims 1-5 and 9-13 of the '428 Patent are unpatentable for the reasons provided below.

A. Qualifications

4. As indicated in my *curriculum vitae* ("CV"), included as Ex-1003, I am currently a Principal Research Scientist at the Massachusetts Institute of Technology ("MIT"). My CV includes additional information about my professional history and

contains further details on my experience, publications, patents, and other qualifications to render an expert opinion. Herein, I highlight experiences relevant to the technology of the patent at issue.

5. I earned a Bachelor of Science in Engineering from Carnegie Mellon University in 1994 and a Master's degree in Engineering from MIT in 1998. My thesis topic related to anisotropic wave guides, image processing, timeseries analysis, applied to non-destructive testing. In 2006, I earned my Ph.D. in Engineering from MIT researching in the AI Lab. My research focused on high-performance computation, signal processing, and video-instrumentation system design, video search, human-interfaces for algorithm interaction, and video compression.

6. In 1997, I co-founded Xcitex Inc., a company that specialized in video-acquisition and motion-analysis software. As CTO and first engineer at Xcitex, I designed and built video analytics and streaming products for scientific markets, including GUIs (graphical user interfaces), compression, and key frame technologies and led team to do the same. I served as the Chief Technology Officer and directed and managed product development until 2006. I served as the technical and scientific advisor to eXstream Solutions, one of the earliest video streaming companies. My work there focused on digital media application development, multimedia content

and rights management, hardware and network architecture, computer interface design, digital design, and consulting.

7. I joined MIT in 2006 and have variously served as the Director of the Master of Engineering in Advance Manufacturing and Design Program, Co-Director of the Medical Electronics Device Realization Center. Many of the courses, projects, and papers my students undertake involve technologies relevant to the '428 Patent, for example human computer interfaces for image display and analysis, software workflow systems, and video and image processing applied to manufacturing system and medical imaging systems.

8. Furthermore, my research focuses on imaging, video processing, human interface systems. I teach or have taught courses in Electrical Engineering, Controls, Optics, Networking Communications, Signal Processing, Video Processing, Imaging, Complex Systems (including hardware and software) Design, all pertinent subject matter to the '428 Patent.

9. For decades I have applied principles of human machine interfaces, video analytics, system design, and compression to domains such as manufacturing imaging, motion capture, and medical imaging. I have extensive experience in video processing, video analytics, graphical user interfaces (GUIs), human-in-the loop workflow design, compression, and key frame detection, and have published widely on video processing across multiple fields. I was awarded an Emmy Award from

the National Academy of Television Arts & Sciences for technical innovation in Broadcast television related to video processing and analysis. I have published approximately 100 papers and have received a number of best paper and distinguished paper awards. I am a co-author of a number of papers that relate to the technology in the ’428 Patent, such as:

- Anthony, Brian W., and Fitriani Chua. “Computationally efficient optimal video comparison for machine monitoring and process control.” Journal of Manufacturing Science and Engineering 139, no. 10 (2017): 101007.

- Sun, Shih-Yu, Matthew Gilbertson, and Brian W. Anthony. Computer-guided ultrasound probe realignment by optical tracking. In 2013 IEEE 10th International Symposium on Biomedical Imaging, pp. 21-24. IEEE, 2013.

- Sun, S.Y., Gilbertson, M. and Anthony, B.W., 2014, September. Probe localization for freehand 3D ultrasound by tracking skin features. In International Conference on Medical Image Computing and Computer-Assisted Intervention (pp. 365-372). Cham: Springer International Publishing.

- Magana-Salgado, Uriel, Praneeth Namburi, Micha Feigin-Almon, Roger Pallares-Lopez, and Brian Anthony. "A comparison of point-tracking algorithms in ultrasound videos from the upper limb." BioMedical Engineering 22, no. 1 (2023): 52.

- Ljubicic, Dean M., and Brian W. Anthony. "3D high-speed profilometer for inspection of micro-manufactured transparent parts." In Optical Measurement Systems for Industrial Inspection VII, vol. 8082, pp. 320-331. SPIE, 2011

B. Materials Considered

10. In forming my opinions, I reviewed the following documents, in addition to others I have cited in my declaration. I also relied on my own knowledge of and experience.

Ex. No.	Description
Ex-1001	U.S. Patent No. 9,235,428 ("the '428 Patent")
Ex-1002	Declaration of Dr. Brian Anthony
Ex-1003	Curriculum Vitae of Dr. Brian Anthony
Ex-1004	Prosecution History of U.S. Patent No. 9,235,428
Ex-1005	U.S. Patent No. 6,886,013 to Beranek ("Beranek")
Ex-1006	U.S. Patent No. 6,188,401 to Pleyer ("Pleyer")
Ex-1007	U.S. Patent No. 6,208,336 to Carter ("Carter")

Ex. No.	Description
Ex-1008	U.S. Patent No. 6,473,407 to Ditmer ("Ditmer")
Ex-1009	M. Satyanarayanan, Jason Flinn & Kevin R. Walker, *Visual Proxy: Exploiting OS Customizations without Application Source Code* (Carnegie Mellon Univ., Sch. of Comput. Sci., July 1, 1999)
Ex-1010	Badriprasad Keshavamurthy, *Software Architecture and Cross Platform User Interfaces* (Master's thesis, Tex. Tech Univ., Aug. 1996)
Ex-1011	Thomas Tilley, *Distributing a Platform Independent Graphical User Interface* (1998)
Ex-1012	David S. Wishart et al., *A platform-independent graphical user interface for SEQSEE and XALIGN*, 13 Comput. Appl. Biosci. 561 (1997)
Ex-1013	Frank Kargl, Torsten Illmann, Michael Weber & Stefan Ribhegge, Dynamic User Interfaces with Java (Distributed Systems Dep't, Univ. of Ulm, n.d.
Ex-1014	INTENTIONALLY LEFT BLANK
Ex-1015	INTENTIONALLY LEFT BLANK
Ex-1016	INTENTIONALLY LEFT BLANK
Ex-1017	INTENTIONALLY LEFT BLANK
Ex-1018	INTENTIONALLY LEFT BLANK
Ex-1019	INTENTIONALLY LEFT BLANK
Ex-1020	Claim Mapping Table
Ex-1021	*Adeia Techs., Inc. v. The Walt Disney Co.*, No. 1:24-cv-01231, Defendants' Supplemental Disclosure of Proposed Claim Constructions (D. Del. Sept. 16, 2025)

Ex. No.	Description
Ex-1022	*Adeia Techs., Inc. v. The Walt Disney Co.*, No. 1:24-cv-01231, Plaintiffs' Initial Proposed Claim Constructions (D. Del. Sept. 5, 2025)

II. LEGAL STANDARDS

11. In forming my opinions and considering the subject matter of the '428 Patent and its claims in light of the prior art, I am relying on certain legal principles that counsel in this case explained to me. My understanding of these concepts is summarized below.

12. I understand that earlier publications and patents may act to render a patent unpatentable for one of two reasons: (1) anticipation, and (2) obviousness.

A. Anticipation

13. As explained to me by counsel, the claims of a patent are anticipated by a prior art reference if each and every element of the claim is found either explicitly or inherently in the reference. I understand that inherency requires a showing that the missing descriptive matter in the claim is necessarily present in the allegedly anticipating reference, and that it would have been so recognized by a person of ordinary skill in the art ("POSITA").

14. I understand that when a challenged claim covers several structures, either generically or as alternatives, the claim is deemed anticipated if any of the structures within the scope of the claim is found in the prior art reference.

15. Although anticipation typically involves the analysis of a single prior art reference, I understand that additional references may be used to show that the prior art reference has enabling disclosure (i.e., allows a POSITA to make the invention without undue experimentation), to explain the meaning of a term used in the prior art reference, and/or to show that a characteristic is inherent in the prior art reference.

B. Obviousness

16. As explained to me by counsel, a claim is invalid as obvious if it would have been obvious to a person of ordinary skill in the art at the time the alleged invention was made. This means that even if all of the elements of the claim cannot be found in a single prior art reference that would anticipate the claim, a person of ordinary skill in the art who was aware of the prior art would have been able to come up with the claimed invention. This may be the case, for example, where the missing element represents only an insubstantial different over the prior art or a reconfiguration of a known system. I understand that in an obviousness determination, the person of ordinary skill in the art is presumed to have knowledge of all material prior art.

17. I understand that an obviousness analysis requires an understanding of the scope and content of the prior art, any differences between the alleged invention and the prior art, and the level of ordinary skill in evaluating the pertinent art.

18. I understand that when a product is available, design incentives and other market forces can prompt variations of it, either in the same field or a different one. If a person of ordinary skill in the art can implement a predictable variation, obviousness likely bars its patentability. For the same reason, if a technique has been used to improve one device and a person of ordinary skill in the art would recognize that it would improve similar devices in the same way, using the technique would have been obvious.

19. I understand that whether a prior art reference renders a patent claim unpatentable as obvious is determine from the perspective of a person of ordinary skill in the art at the time of the alleged invention. I have been told that there is not requirement that the prior art contain an express suggestion to combine known elements to achieve the claimed invention, but a suggestion to combine known elements to achieve the claimed invention may come from the prior art, as filtered through the knowledge of one skilled in the art. In addition, I have been told that the inferences and creative steps a person of ordinary skill in the art would employ are relevant to the determination of obviousness.

20. I understand that one may consider, e.g., whether (1) the change was merely the predictable result of using prior art elements according to their known functions, or whether it was the result of true inventiveness; (2) there is some teaching or suggestion in the prior art to make the modification or combination of

elements claimed in the patent; (3) the claimed innovation applies a known technique that had been used to improve a similar device or method in a similar way; (4) the claimed invention would have been obvious to try, meaning that the claimed innovation was one of a relatively small number of possible approaches to the problem with a reasonable expectation of success by those skilled in the art; (5) the invention merely substituted one known element for another known element in order to obtain predictable results; (6) the invention merely applies a known technique to a known device, method, or product to yield predictable results; or (7) known work in one field of endeavor may have prompted variations of it for use in either the same field or a different one based on design incentives or other market forces that would have been predictable to a person of ordinary skill in the art.

21. I further understand that certain factors may support or rebut the obviousness of a claim. I understand that such secondary considerations include, among other things, commercial success of the patented invention, skepticism of those having ordinary skill in the art at the time of the invention, unexpected results of the invention, any long-felt but unsolved need in the art that was satisfied by the alleged invention, the failure of others to make the alleged invention, praise of the alleged invention by those having ordinary skill in the art, and copying of the alleged invention by others in the field. I understand that there must be a nexus—that is, a connection—between any such secondary considerations and the alleged invention.

I also understand that contemporaneous and independent invention by others is a secondary consideration tending to show obviousness.

22. I am not aware of any allegations by the named inventor of the '428 Patent or any assignee of the '428 Patent that any secondary considerations tend to rebut the obviousness of the '428 Patent.

23. Additionally, I understand that in considering obviousness, it is important not to use the benefit of hindsight derived from the patent under consideration.

C. Level of Ordinary Skill in the Art

24. When interpreting a patent, I understand that it is important to identify the relevant art pertaining to that patent, as well as the level of ordinary skill in that art at the time of the claimed invention. The "art" is the field of technology to which the patent is related.

25. I am informed and understand that the POSITA is a hypothetical person who is presumed to know the relevant prior art. I understand that the actual inventor's skill is not determinative of the level of ordinary skill. I further understand that factors that may be considered in determining level of skill include: (i) the types of problems encountered in the art; (ii) prior art solutions to those problems; (iii) the rapidity with which innovations are made; (iv) the sophistication of the technology; and (v) the educational level of active workers in the field. I

understand that not all such factors may be present in every case, and one or more of them may predominate.

26. I understand that a POSITA is one who is presumed to be aware of all pertinent art, thinks along conventional wisdom in the art, and is a person of ordinary creativity.

27. A person of ordinary skill in the art at the relevant time, which I have been told to consider as February 1, 1999 ("POSITA") would have had at least a four-year degree in electrical engineering, computer science, computer engineering, or a related field, and two-to-three years of experience in the research, design, development, or testing of interactive interface and display technologies, user interfaces, and platforms, with additional education substituting for experience and vice versa. .

28. Based on my knowledge, skill, and experience, I have an understanding of the capabilities of a person of ordinary skill in the relevant art. For example, from my industry experience, I am familiar with what an engineer designing graphical user interfaces, computer based image and information interactive-displays, and video-acquisition and motion-analysis software would have known and found predictable in the art. From teaching and supervising my post-graduate students, I also have an understanding of the knowledge that a person with this academic

experience possesses. Furthermore, I possessed the knowledge of a POSITA myself at least as of 1997.

D. Claim Construction

29. I understand that the United States Patent and Trademark Office interprets claim terms in an *inter partes* review proceeding under the same claim construction standard that is used in a United States federal court. I understand that under this standard, the meaning of claim terms is considered from the viewpoint of a POSITA at the time of the alleged invention.

30. I have been informed that claim terms are generally given their ordinary and customary meaning as understood by one of ordinary skill in the art in light of the specification and the prosecution history pertaining to the patent. I understand, however, that claims terms are generally not limited by the embodiments described in the specification.

31. I understand that in addition to the claims, specification, and prosecution history, other evidence may be considered to ascertain the meaning of claim terms, including textbooks, encyclopedias, articles, and dictionaries. I have been informed that this other evidence is often less significant and less reliable than the claims, specification, and prosecution history.

III. SUMMARY OF GROUNDS

32. I understand that Petitioner challenges the validity of Claims 1-5 and 9-13 of the ’428 Patent. It is my opinion that the following grounds demonstrate that the claims of the ’428 Patent are invalid.

Ground	Summary
1	Claims 1-5 and 9-13 are obvious over Beranek (Ex-1005) in view of Pleyer (Ex-1006)
2	Claims 3 and 11 are obvious over Beranek-Pleyer in view of Carter (Ex-1007)
3	Claims 9-11 are obvious over Ditmer (Ex-1008) in view of Beranek

IV. THE ’428 PATENT

A. Overview of the ’428 Patent

33. The ’428 Patent relates to a computer implemented system that provides a “graphical user interface (GUI) for an application program” using a “renderer, a proxy and a workflow manager.” Ex-1001, Abstract.

34. As shown below in Figure 7[1], renderer 200 (green) communicates with an application proxy 205 (purple) and receives “instantiated user interface components 208” and “instantiated document viewer 212” (blue) to render the

[1] Throughout, emphasis and annotations are added unless otherwise specified.

graphical user interface (GUI). Ex-1001, 4:7-10, 5:17-18. User interface components are "controls such as button, sliders, radio buttons, edit boxes, etc.," that can be implemented using "ActiveX controls, Java applets, etc." *Id.*, 5:30-34. Document viewers include "viewers appropriate to observe and interact with data appropriate to the application program, in all of its possible contents." *Id.*, 5:34-37.

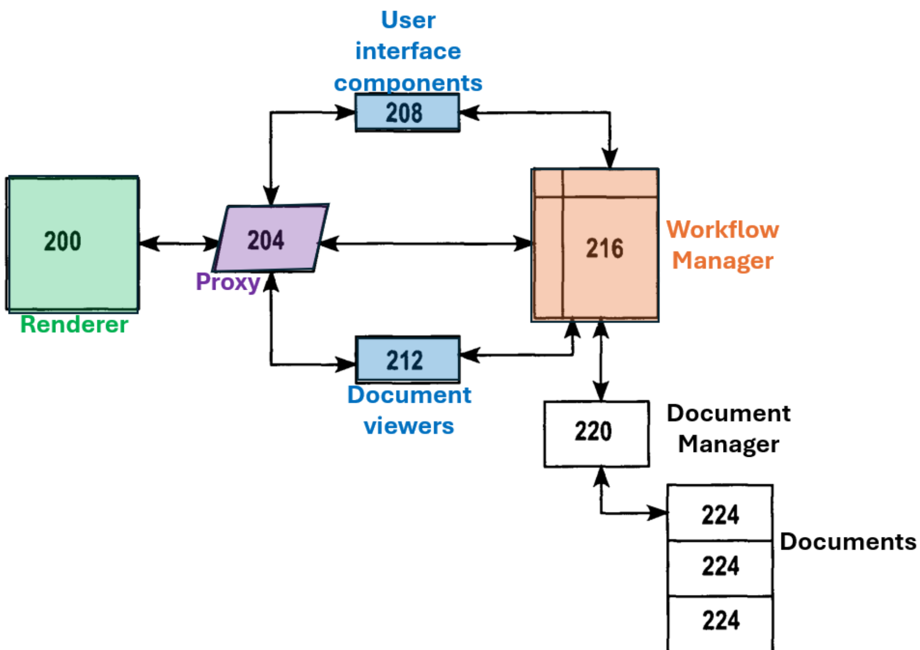


FIG. 7

35. Proxy 204 appears "as a single, monolithic application program and all details of the components which make up the application program are essentially hidden from renderer 200 which need only interface to application proxy 204." Ex-1001, 5:8-13. In response to a "context change, or other event requiring a display update occurs within the application program, either as a result of a user interface event or another event," proxy 204 determines the changes required to be made to

the application program user interface, including rendering a new layout or updating the present layout. *Id*., 6:55-66. The proxy sends a message to the renderer 200, which will then re-render the application program user interface to dynamically update the display to reflect the changes. *Id*. Application proxy 204 also communicates to a "workflow manager 216" (orange) for the updated elements to be rendered. *Id*., 4:30-34, 6:66-7:5.

36. As explained below, all subject matter of the challenged claims is described in the prior art and the '428 claims are obvious in view of the prior art.

B. Prosecution History of the '428 Patent

37. The '428 Patent was filed August 23, 2006, as a continuation of Application No. 09/240,844, filed on February 1, 1999, now Pat. No. 7,117,446. Ex-1001. For purposes of my analysis, I applied the earliest alleged priority date of the '428 Patent: February 1, 1999.

38. The '428 Patent was allowed only after a single round of examination and an appeal to the Board. The Examiner rejected the claims under 35 U.S.C. §§ 101, 103. In response, the Applicant argued that U.S. Patent No. 6,038,395 to Chow et al. ("Chow") did not teach a "single monolithic application program" because Chow's proxy manipulates and edits the proxy objects "at build time," and that the claims require the proxy "to determine changes required to be made to the GUI for the application program during execution of the application program." Ex-1004, 83.

The Examiner responded that Chow "discloses that Interface repository being contained in the proxy object itself therefore is a monolithic application and there could only one proxy object," and again rejected the claims under 35 U.S.C. § 103. The Examiner further added U.S. Patent No. 7,168,086 to Carpenter to the Lynch-Chow combination for its teaching of a proxy server that determines the user interface elements to be presented on the client terminal's viewer. *Id*., 93-94.

39. The Applicant subsequently appealed the Examiner's rejection of the recited proxy and workflow manager to the Board. Specifically, Applicant argued that Chow's proxy is not a "single monolithic application program" because the proxy would have been understood as using "multiple proxy objects, where each proxy object corresponds to one target object." Ex-1004, 127-130. Applicant also argued that Lynch did not teach the "workflow manager" because it merely teaches a "formatter 707, which "insert[s] proper formats, such as line breaks," and Applicant could not discern how the surface formatting suggests the recited workflow manager that communicates document viewers and user interface components. Ex-1004, 130.

40. On May 5, 2015, the Board reversed the Examiner's rejection, finding that the Examiner had not adequately identified a "workflow manager" in the primary reference, Lynch. Ex-1004, 189-190. The '428 Patent was allowed based on

Examiner's amendment to overcome the pending § 101 rejection. Ex-1004, 200; *see also id.*, 248 (granting after Applicant's minor amendment).

41. During prosecution, the Examiner listed nineteen references on its Notice of References cited forms. *See* Ex-1004, 58-59, 101-102. U.S. Patent No. 6,473,407 to Ditmer et al. was listed but never applied to the claims. *Id.*, 58, 101.

42. The Examiner materially erred by not considering the Ditmer during prosecution because, as demonstrated in Ground 3 below, Ditmer, in view of Beranek, respectively, renders obvious each limitation of Claims 9-11. It was a material error to not apply Ditmer's teachings to the amended claims during prosecution.

43. In addition to the Beranek-Pleyer and Beranek-Pleyer-Carter grounds, the prior art I apply to the claims in my Declaration includes new arguments and prior art not before the USPTO during prosecution of the '428 Patent. And as I discuss below, this art discloses the '428 Patent's Claims 1-5 and 9-13, including the features the applicant contended were new during prosecution.

V. TECHNOLOGY BACKGROUND

A. Consistent presentation of Web documents was a well known GUI problem and solution prior to 1999

44. By the turn of the century, POSITAs recognized the drawbacks of fixed, hard-coded interfaces that could not be updated without recompiling or reinstalling, and they pursued run-time reconfiguration to expose new features without

redeployment. For example, U.S. Patent No. 6,208,336 to Carter et al., filed on March 20, 1998, and issued on March 27, 2001 ("Carter," Ex-1007), teaches "dynamically construct[ing] a graphical user interface associated with an application" by discovering which command classes can be instantiated and then "construct[ing] the graphical user interface such that the graphical user interface contains the additional functions of the successfully instantiated command classes." Ex-1007, 2:14–23. As shown below in Figures 5A and 5B, Carter shows controls before and controls added at run-time (e.g., "Spell Checker" and "Thesaurus"), confirming the field's focus on adding capabilities without rebuilding the application.

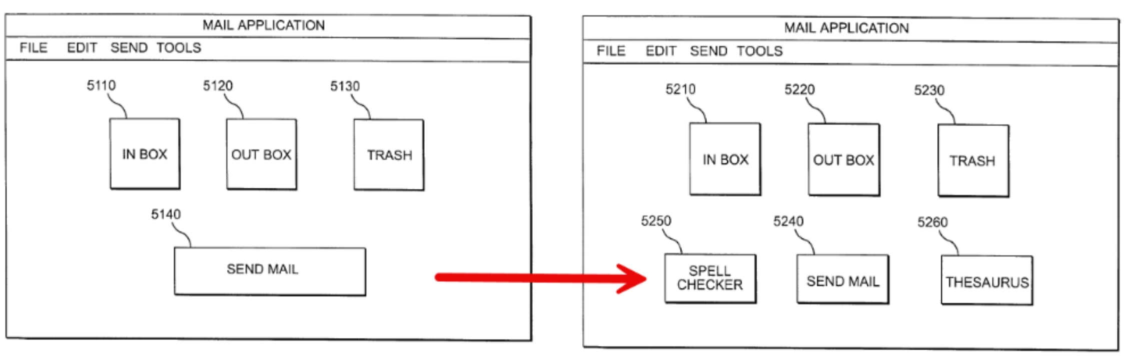


FIG. 5A *FIG. 5B*

45. Carter noted that POSITAs "have realized the advantages of using modular concepts in the design of computer program logic," because software built using "modular concepts are more portable to other hardware platforms, more compatible with other software and hardware interfaces, and tend to be more easily

designed, tested, installed, and maintained." Ex-1007, 1:9-14. Carter recognized that creators were developing separate versions of their applications in order to provide additional functions that the customers desire but providing the additional functions only when those functionalities are needed. *Id*., 1:54-58. Carter also noted that the alternative, of making only a single version integrating all of the additional functionality, increases cost of software production and forces customers that desire only the basic application to purchase additional functionality. *Id*., 1:59-67. Carter noted the need to provide customers with an application having only the desired functionality without requiring the creation of several versions of the application, by "determining at run-time what functionality is available, and modifying the graphical user interface associated with the application to display controls for the available functions." *Id*., 2:9-13.

46. Other independent literature of the time also documented front-end interposition to extend existing GUIs at run time. For example, in 1999, Mahadev Satyanarayanan, Jason Flinn, Kevin R. Walker published an article titled "Visual Proxy: Exploiting OS Customizations without Application Source Code." *See* Ex-1009 ("Satyanarayanan"). In the article, Satyanarayanan described that "[p]erformance and functionality enhancements are often made available to applications through extensions to an operating system's API." Ex-1009, 1. APIs are used for "search latency reduction, application-specific conflict resolution, mobility,

informed prefetching, consistency management, and low-overhead transactions." *Id*., 1 (internal citations omitted). Satyanarayanan discussed how "source code to some of the most popular and important applications is often not available under acceptable licensing terms." *Id*. Satyanarayanan proposed a "solution… that appears promising for a broad class of interactive applications that rely on graphical user interfaces (GUI)," by using a "visual proxy" interposed in the window system channel can "snoop on changes to the GUI" and "generate synthetic window events," augmenting an application's controls without modifying its source or renderer. Ex-1009, 1-2. The visual proxy further "creates a separate GUI" and seamlessly integrates it with the application via reparenting to reflect new capabilities—all without source code changes. *Id.*, 2. Satyanarayanan explained how the "concept of a proxy is well-established… in Web browsers" and are "easy to implement." *Id*., 1. Indeed, in the Web context, the use of a proxy was already baseline practice, with contemporaneous work describing HTTP proxies that intercept, mediate, and transform client/server exchanges. *See generally* Ex-1010 (B. Keshavamurthy, Software Architecture Across Platform User Interfaces, at ch. 2).

47. Similar systems used supervisory logic and scripting to create and close dialogs and controls during run time. For example, U.S. Patent No. 6,188,401 to Pleyer ("Pleyer," Ex-1006), filed on March 25, 1998, and issued on February 13, 2001, explains that "controls and dialogs… are implemented as text and graphics

documents written in … HTML" (Ex-1006, 7:39–42) and that a "global script file" uses open/close methods such as ShowModalDialog and ShowModelessDialog to present and dismiss UI components, (*id*., 8:56–60) even "creat[ing] a new user interface component in response to a user pressing a specific key." *Id*., 9:49–52.

48. U.S. Patent No. 6,473,407 to Ditmer ("Ditmer," Ex-1008), filed on September 24, 1998, and issued on October 29, 2002, also teaches a web-based proxy layer that "sends the generic header followed by the proxy-specific response back … to the user browser," (Ex-1008, 14:43–46) while a distinct "event monitor server is responsible for … data collection, calculation, storage, and report generation," (*id*., 14:60–65) illustrating the separation of upstream decision-making from rendering and the use of a façade to mask internal complexity behind a unified interface. Both Pleyer and Ditmer are examples of how an upstream proxy decides presentation while the renderer remains blind to componentization, whether the interposed layer operates at the HTTP stream or at the GUI event channel. In both cases, the renderer receives only the formatted output to be displayed, such that the renderer is not aware of the details of the formatted output.

49. As Carter noted, POSITAs were concerned with providing the additional functionality to users without making separate versions of varying functionality. U.S. Patent No. 6,886,013 to Beranek ("Beranek," Ex-1005), filed on August 20, 1997, issued on April 27, 2004, is another example explaining this

problem. Beranek describes that with the increase of using the world wide web for multimedia information, HTML documents are provided. HTML "provides basic document formatting and allows the developer to specify 'links' to other servers and files." Ex-1005, 1:13-30. Beranek describes that "Web page authors design their Web content using standard design constraints and HTML formatting constructs" but even then, "it is often the case that a given Web page looks different when viewed on different client machines." *Id*., 1:46-52. Because of "the differences between browser applications, web designers must often design multiple versions of the same or similar content in order to attempt to maintain consistent content display across different browser types." *Id*., 1:53-56. Yet multiple versions must then be supported on a given server, which slows page access time and leads to redundant or wasted Web site storage capacity. *Id*., 1:56-59. Therefore, Beranek notes the "problem of presenting Web content in a consistent manner across multiple display system formats is exacerbated as more and more machines … are provided with the capability of displaying such content." *Id*., 1:60-67.

50. Beranek provides a "mechanism for processing a Web document such as is retrieved from a Web server in order to modify its display characteristics in a dynamic manner before delivery to the browser," which obviates "generation and storage of multiple versions of a particular Web Page yet ensure that the page is displayed consistently across multiple Web content display system types." Ex-1005,

2:1-10. Beranek describes a client-side proxy that intercepts a data stream and "inject[s] new control information into the data stream in order to affect how the Web content is ultimately displayed on the browser," (*id*., 13:11–13) after which the "reformatted Web document is then passed to the browser for display." *Id*., 11:47–49. These upstream selection and formatting decisions, made prior to rendering, yielded consistent presentation while keeping the renderer oblivious to internal mechanics. Therefore, POSITAs were aware of the inconsistent look-and-feel across browsers, operating systems, and devices and prescribed proxy-side normalization and provided solutions prior to 1999.

B. Platform-independent renderers and dynamic updates

51. By the 1999, browsers (HTML/DOM/CSS and DHTML) and Java/Swing were widely used as platform-independent renderers capable of dynamic layout and widget manipulation at run-time. Pleyer (discussed above) explains that the "user interface is implemented using individual HTML … components in conjunction with a global Script written in a … language such as JavaScript," and that the script "responds to user input" by opening and closing dialogs and updating components. Ex-1006, Abstract, 2:1–10. JavaScript "allows a designer to add interactivity to HTML, to allow for user interaction and feedback, multimedia and animation, and to link HTML to other technologies such as Java, ActiveX, and plug-ins," reflecting a standard technique to render a new layout or update a present

layout during execution without changing the underlying application binary. *Id.*, 4:11-15.

52. Thomas Tilley also described this in an article titled "Distributing a Platform Independent Graphical User Interface." *See* Ex-1011 ("Tilley"). Tilley discusses how Java is an "language that was designed to be platform independent, web aware, and secure." *Id.*, 0017-0018. Specifically, Java's platform independence "is essentially the ability to move software between different operating systems without re-compilation." *Id.*, 0018. A Java program needs only be compiled once. *Id*. Java Applets are small applications that can be embedded in HTML pages and be accessed via the Web and run using a compatible web-browser. *Id*. Indeed, standards contemporaneous with the priority date (e.g., HTML 4.0 and DOM Level 1) expressly defined a script-addressable document model for run-time structure and presentation updates, and Java/Swing's pluggable look-and-feel (PLAF) and component model provided the same cross-platform rendering independence on the desktop. *See* Ex-1011.

53. Tilley and other academic and industrial work likewise emphasized platform-independent rendering and remote UI assembly. Tilley specifically describes "a platform-independent graphical user interface framework" that "dynamically build[s] an application's existing interface on a remote client," so that "changes to the application do not necessitate changes to the remote framework" and

"the remote interface will automatically update itself to reflect those changes." Ex-1011, 0002. Tilley further stresses preserving a consistent "look- and-feel" while enabling updates at run-time, further underscoring the routine nature of dynamic, renderer-agnostic UI composition. *Id.* Similar platform-independent GUIs in other domains (e.g., Smalltalk-based scientific tools wrapping C back-ends) likewise separated front-end viewers from back-end computation across heterogeneous systems, reporting GUIs that "look and operate almost identically across all major platforms" while preserving C back-end code via well-defined calls. *See, e.g.*, Ex-1012 (Wishart et al, *A platform-independent graphical user interface for SEQSEE and XALIGN*, 561).

54. In summary, POSITAs were implementing browsers as platform-independent renderers capable of dynamic layout and widget manipulation at run time.

C. It was well known for proxies and coordination layers to make presentation decisions while the renderer remained blind.

55. It was common to place a proxy or coordination layer between application logic and the renderer to standardize layout, inject control information, and conceal component details. For example, Beranek's proxy "parsing the Web document to identify formatting information" and "altering the formatting information" before "passing the Web document to the browser" exemplifies this pattern. Ex-1005, cl. 1. Ditmer similarly teaches a proxy that packages and returns

responses to the browser while upstream services perform data collection and report generation; as the reference puts it, the proxy "sends the generic header followed by the proxy-specific response back … to the user browser," masking upstream orchestration behind a unified interface. Ex-1008, 14:40–46. This was also formalized at the protocol level: HTTP/1.1 recognized proxies as first-class intermediaries that receive, parse, transform, and forward responses, so that from the renderer's perspective the proxy appears as the single source. Contemporaneous implementations expressly targeted compliance with HTTP/1.1 (RFC 2068) while interposing a filtering proxy ahead of the browser. Ex-1005, 8:58–62.

56. At the GUI boundary, contemporaneous "visual proxy" techniques demonstrate the same supervisory–renderer separation by interposing a front-end layer that is "located at the front end of a GUI-based application and modifies its visual appearance," while leaving the underlying application unmodified. Ex-1009, 1–2 (Satyanarayanan, *Visual Proxy* 1). A visual proxy "interposed in this channel can snoop on changes to the GUI" between the application and window system, and it "can generate synthetic window events simulating user input," thereby "augment[ing] the GUI" and driving application behavior via standard event delivery. In practice, the proxy "creates a separate GUI" and then "seamlessly integrates this interface with the application using a technique called reparenting," so it can transparently inject additional interface elements. *Id.* For example, the

proxy "inserts an additional menu-bar" and "displays a set dialog box" in Netscape, and adds "new buttons" to FrameMaker—all shown as being achieved "without [access to] application source code." *Id.* Because the added components are reparented into the existing window tree, the renderer remains unaware of the provenance or internal structure of the injected UI parts; from its perspective, the composite interface is indistinguishable from one produced by the application itself. This façade-style interposition thus enables transparent modification and extension of the user interface at runtime—e.g., dynamically adding menus or pop-up dialogs, overlaying controls, or redirecting command invocation—while preserving binary compatibility and leaving both the application code and rendering engine unmodified.

D. The architecture's language- and stack-independence was well understood

57. The renderer–proxy–workflow split was expressly implementation-agnostic, spanning DHTML/HTML/JavaScript, COM/ActiveX embedded in HTML, Java applets/Beans, and all-Java (AWT/Swing) stacks. Contemporary sources observed that mixed-technology viewers and components were routinely composed while the proxy/façade and supervisory roles remained constant. Ex-1006, 4:11–15, 2:1–10. Pleyer (discussed above) teaches that the global script and HTML components "expose methods and properties" that allow them to interoperate, (Ex-1006, 6:26–35) while the supervisory layer determines

which components are rendered. *Id.*, 7:39–49. Beranek (discussed above) also teaches a proxy that can "change … font types and size, to control display geometry, … [and] add[,]remove[, or]reconfigure frames," irrespective of the underlying platform or browser variant. Ex-1005, 10:30-40.

58. Engineering work on distributed and remote UI frameworks reinforced that upstream decision-making governed what the renderer displayed, independent of the renderer's implementation details, and that presentation changes could be delivered at run-time without modifying the client-side rendering engine. *See* Ex-1011, 0002 (Tilley at Abstract); Ex-1013. Long-standing window-system models such as X11 further demonstrate the split between an upstream application client and a display server (renderer): "The application can be considered as the X-client and the GUI the X-server. The X-Server renders the GUI on the user's display and handles input events." Ex-1011, 0011-0012 (Tilley at numbered pages 2–3). Cross-platform toolkits (e.g., Qt) and Swing's pluggable look-and-feel likewise show the same supervisory–renderer pattern across stacks. *Id.*, 0025-0026 (Tilly at numbered pages 16–17). Additionally, Frank Kargl, et al. published an article titled, "Dynamic user interfaces with Java," which explains using applets so that "new agents" are able to join a distributed infrastructure "in a dynamic manner without need to change the framework" and without any "built-in restrictions to the complexity of the user interface." Ex-1013, Abstract. Kargl discusses the use of a

User Communication Agent (UCA), that handles display of the GUI integrated into the web browser. *Id*., 2. As shown below in Figure 1, the UCA is "partitioned into a generic Java Applet that runs in the user's web browser and a server module that connects to the Agent Bus and handles communication with other agents." *Id*. The UCA allows users to create, modify and transfer components. *Id*., 4. Kargl notes that this allows user interfaces to adapt to a wide range of possible displays like PCs, PDAs, "or possibly even voice control." Therefore, Kargl is another example of proxy and supervisory manager performing functions without the renderer's awareness.

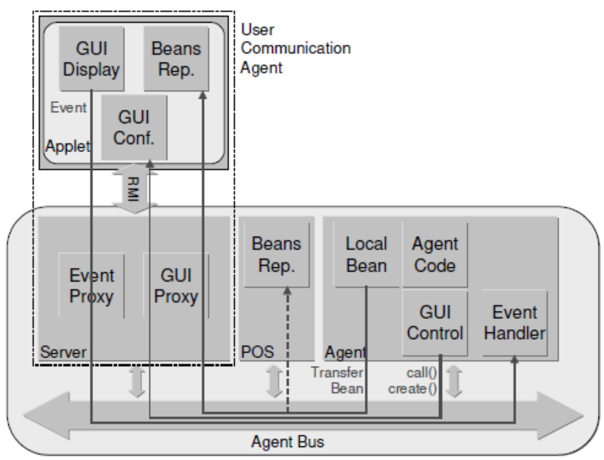


Figure 1: System Architecture

59. In summary before 1999, the field had already converged on the same architectural concepts at issue here: use a platform-independent renderer; interpose a proxy/façade that intercepts and reformats content; employ a supervisory workflow to select and communicate document viewers and UI components; and keep the renderer blind to upstream composition so components can be added, substituted, or

reorganized dynamically. As the references put it, proxies "inject new control information" (Ex-1005, Beranek, 13:9–14) and "send … the proxy-specific response back … to the user browser," (Ex-1008, Ditmer, 14:41–44) supervisory scripts "generate[] dialog boxes and controls," (Ex-1006, Pleyer, 7:39–42) and dynamic GUI composition "contains the additional functions of the successfully instantiated command classes"—all without recompiling the application or modifying the renderer (Ex-1007, Carter, Abstract).

VI. BRIEF DESCRIPTION OF THE APPLIED PRIOR ART

A. Beranek (Ex-1005)

60. Beranek (U.S. Patent No. 6,886,013) was filed on August 20, 1997, issued on April 27, 2004. As explained to me by counsel, I understand that Beranek is prior art to the '428 Patent under at least 35 U.S.C. § 102(e).

61. Beranek teaches "[a] method of controlling presentation on a client of a Web document formatted according to a markup language and supported on a server, the client including a browser and connectable to the server via a computer network." Ex-1005, 15:33-36 (Cl. 1). As shown in Figure 9 below, a browser "sits on a given machine (e.g., a personal computer having its own display system)". Ex-1005, 13:2-6. Beranek teaches that a "reformatted Web document is then passed to the browser for display." Ex-1005, 3:7-8, Abstract.

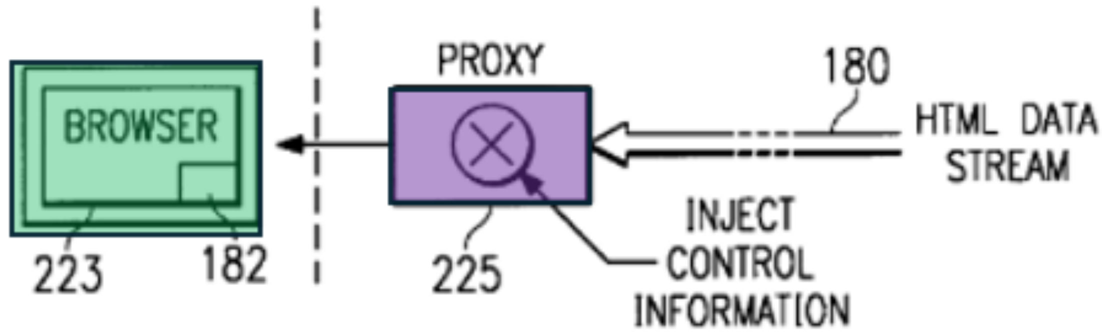


FIG. 9

62. Beranek teaches a "client-side caching proxy," which "intercept[s] the Web document and then dynamically rewrite[s] the document before it is displayed on the browser." Ex-1005, 2:63-3:2. This enables the "proxy to dynamically adjust the 'look and feel' of a given Web document at the client side of the connection." *Id*., 8:39-42. The proxy "is implemented in software and includes a cache 227." *Id*., 8:24-26. Specifically, proxy 225 (purple), intercepts the data stream and "inject[s] new control information … in order to affect how the Web content is ultimately displayed." *Id*., 13:9-14. The proxy is used to modify and/or process data before it is displayed on the user's interface, such that the "data font or presentation to be modified, supplemented or otherwise adapted to the particular display environment." *Id*., 12:17-19. Beranek's proxy thus provides a tool to enhance and tailor the "look and feel" of data displayed and enables the "user interface to be tailored to the Web appliance environment." Ex-1005, 12:19-23.

B. Pleyer (Ex-1006)

63. Pleyer (U.S. Patent No. 6,188,401) was filed on March 25, 1998, and issued on February 13, 2001. As explained to me by counsel, I understand that Pleyer qualifies prior art to the '428 Patent under 35 U.S.C. § 102(e).

64. Pleyer teaches a script-based user interface implementation defining components using a text markup language. Ex-1006, Title. Pleyer teaches a browser program, as shown in Figure 4 below, which includes a user interface component 150 that renders a variable message, scroll indicators, and a connection status area, an internet component 152 that renders internet web content, a television component 154 that renders a live video stream. Ex-1006, 6:40-46, 6:50-51, 6:60-65. Such user interface controls and dialogs are "implemented as HTML files or resources" with the overall "logic and sequence of the user interface" being "established by a global script file written in a scripting language such as JavaScript." *Id.*, 7:18-23.

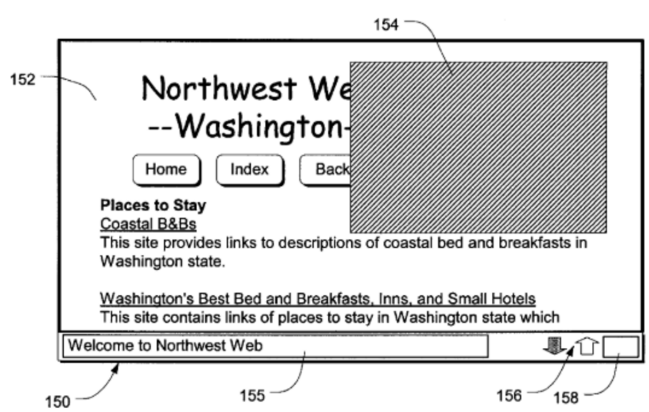


Fig. 4

65. As shown below in Figure 3, Pleyer teaches a supervisory application 100 that "communicates with hardware devices such as the network interface television tuner, keyboard, and display components though conventional Windows operating system calls." Ex-1006, 5:55-65. The supervisory application communicates the global script 106, which includes browsable content 104 and UI components 105. Pleyer teaches that "controls and dialogs… are implemented as text and graphics documents written in … HTML," and that in order to present options to a user, "global script 106 generates dialog boxes and controls by specifying local HTML graphical elements." *Id.*, 7:40-45. JavaScript is used to "add interactivity to HTML, to allow for user interaction and feedback, multimedia and animation, and to link HTML to other technologies such as Java, Active X, and plug-ins." *Id.*, 4:12-16. For example, Pleyer teaches that "event handlers implemented by global script 106" includes generating an "on-screen keyboard" and "audio rendering" (*id.*, 9:1-3, 9:29-30), and "create[ing] a new user interface component in response to a user pressing a specific key." *Id.*, 9:49-51.

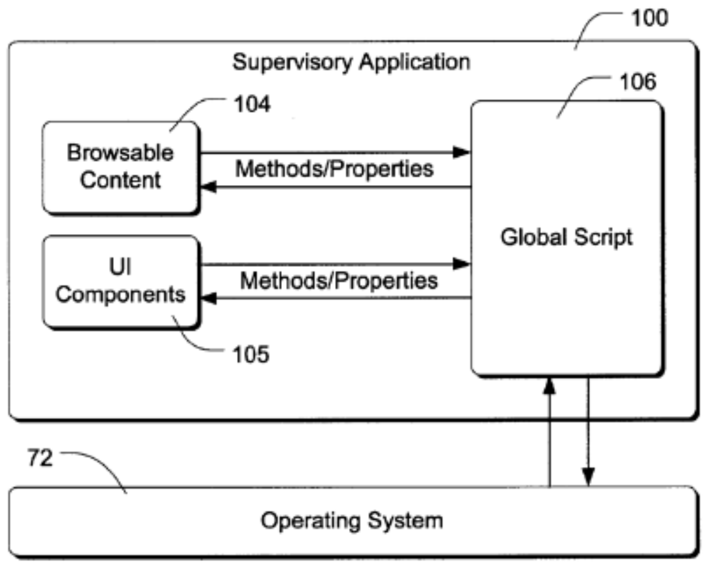


Fig. 3

C. Carter (Ex-1007)

66. Carter (U.S. Patent No. 6,208,336) was filed on March 20, 1998, and issued on March 27, 2001. As explained to me by counsel, I understand that Carter qualifies prior art to the ’428 Patent under 35 U.S.C. § 102(e).

67. Carter teaches a “dynamic graphical user interface feature-set configuration.” Ex-1007, Title. Carter teaches determining which command classes to “instantiate” and constructing “the graphical user interface for the application such that the graphical user interface contains the additional functions of the successfully instantiated command classes.” Ex-1007, Abstract. Carter teaches a constructor 3120 which “locates any packages (or classes in a package).” Ex-1007, 5:27-37. The program thus determines “which classes are available and then construct a graphical user interface ***to reflect all of the classes***.” *Id.*, 2:56-57. Figure

3 shows a command class library which stores multiple command classes 3100-3400, which contain additional functionality of an application. *Id.*, 4:27-30.

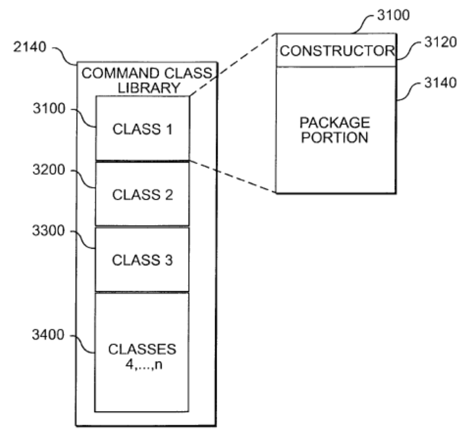


FIG. 3

68. Carter's command classes are rendered on the application so that the user may select the additional functionality, which then causes a new layout in the GUI and updating a present layout in the GUI for the user. *See, e.g.*, Figures 5A, 5B. In Figure 5B, a GUI presents four basic controls of "In Box 5110, Out Box 5120, Trash 5130, and Send Mail 5140" and additional controls for a mail application, by instantiating one or more command classes, such as "a Spell Checker 5250 and a Thesaurus 5260." *Id.*, 5:47-56. Carter's proxy application program components are thus visible to the renderer because the available options/classes are transparent to the renderer and can then displayed to be selected.

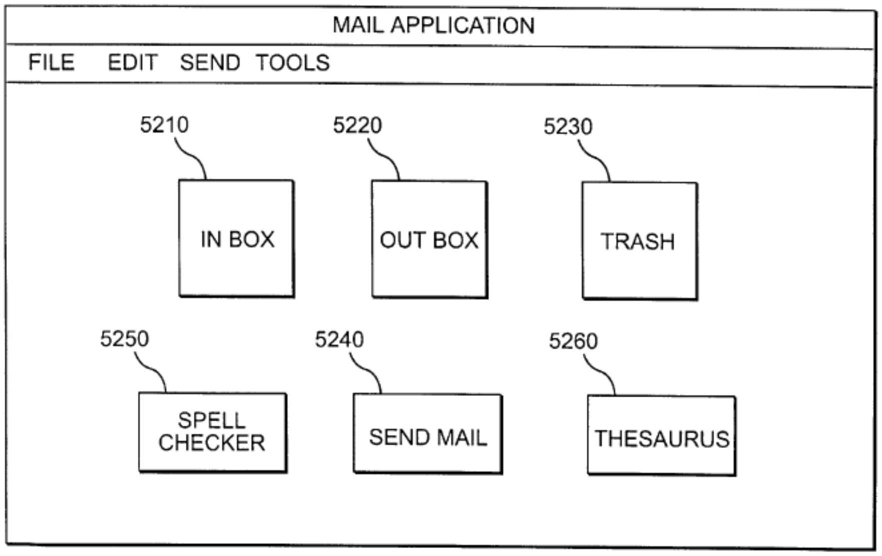


FIG. 5B

D. Ditmer (Ex-1008)

69. Ditmer (U.S. Patent No. 6,473,407) was filed on September 24, 1998, and issued on October 29, 2002. As explained to me by counsel, I understand that Ditmer qualifies prior art to the ’428 Patent under 35 U.S.C. § 102(e).

70. Ditmer teaches a “Web-based fault and alarm management tool that enables customers to monitor and analyze the performance of their voice and data network via a graphical user interface.” Ex-1008, Abstract; *see also id.*, 3:27-36. Ditmer provides a “system and a method for communicating information relating to the dedicated voice and data networks, from an enterprise to a customer at a client workstation, the present invention includes a client browser application located at the client workstation.” *Id.*, 3:52-56. Ditmer states that customer browser is enabled

and includes "***client applications responsible for presentation*** and front-end services." *Id.*, 6:14-16.

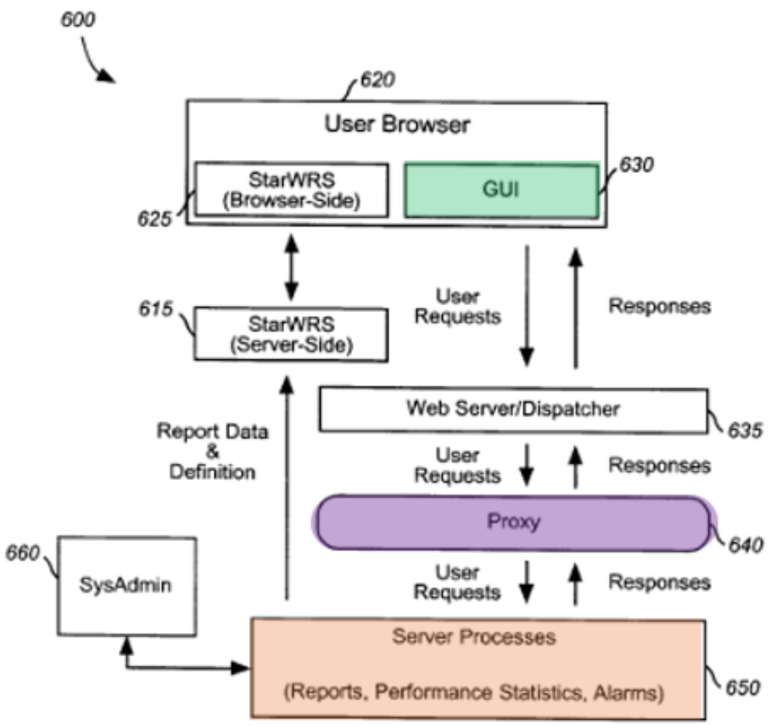


FIG. 6

71. Ditmer teaches an "event monitor system," which includes a three-tier architecture. As shown in Figure 6, first, GUI 630 (green) allows a user to submit requests and receive responses. Second, Ditmer teaches that a user's request is forwarded to "an integrated proxy application 640 [(purple)] for receiving and interpreting the user messages and enabling the event monitor functionality." Ex-1008, 14:31-34. Specifically, the proxy examines and confirms incoming requests to ensure that "they include validly formatted messages for the service with acceptable parameters; a translation process for translating a message into the

database query message or networking protocol; and, a management process for managing the communication of the specific customer request with the middle-tier server to actually get the request serviced." *Id*., 23:7-14; *see also id*., 24:14-20. The "[d]ata returned from the … server is translated back to client format, if necessary, and returned to the dispatch server as a response to the request." *Id*., 23:14-17; *see also id*., 9:46-50. While shown as a separate module, the proxy "may reside on the dispatch server 635 itself." Ex-1008, 24:2-4. Third, Ditmer's "event monitor server 650 [(orange)] is responsible for all tasks leading up to and including the management of alarms and performance reports including data collection, calculation, storage, and report generation." Ex-1008, 14:58-65.

VII. CLAIM CONSTRUCTION

72. For the purposes of my Declaration, I do not believe that any term requires explicit construction to resolve the issues I present in my Declaration because the prior art discloses or renders obvious the claim limitations under any potential construction of the claims.

73. I understand that the Patent Owner and Petitioner in the related litigation have argued the following the claim constructions:

A. "a renderer to render a graphical user interface (GUI) for an application program" (Claim 9)

74. I understand from counsel that, in the related litigation, Petitioner proposed that this term is a means-plus-function elements under 35 U.S.C. § 112 ¶6.

Function: "to render a graphical user interface (GUI) for an application program"

Structure: Insufficient structure; indefinite. Ex-1021.

75. PO has proposed that this limitation should be given a plain and ordinary meaning. Ex-1022. It is my opinion that there is no meaningful difference between these constructions as applied to the prior art references at issue in my declaration. I have applied this limitation to include software that renders a graphical user interface for an application program in the same way as described by the patent, such as with "a DHTML browser." Ex-1001, 4:53, 5:23-24, 7:19.

B. "single monolithic application program" (Claim 9)

76. I understand from counsel that, in the related litigation, Petitioner proposed that the "single monolithic application program" recited in independent claim 9 means a "single self-contained application program." Ex-1021. PO proposes that "the proxy appears to the renderer as a single monolithic application program" means "details of the components that make up the application program are hidden from the renderer." Ex-1022.

77. It is my opinion that there is no meaningful difference between these constructions as applied to the prior art references at issue in my declaration. I have applied this limitation to include this limitation to include software that operates in the same way as "application proxy 204," which is explained to be "a single,

monolithic application program and all details of the components which make up the application program are essentially hidden from renderer 200 which need only interface to application proxy 204." Ex-1001, 5:8-12

C. "the ox" (Claim 9)

78. I understand from counsel that, in the related litigation, Petitioner proposed that the recited "ox" is not amendable for construction. *See* Ex-1021. PO has proposed that "the ox" is "the proxy." *See* Ex-1022. I have applied this term as if the incomprehensible error includes "the proxy."

D. "a workflow manager to communicate one or more document viewers and associated user interface components to the proxy" (Claim 9)

79. I understand from counsel that, in the related litigation, Petitioner proposed that this term is a means-plus-function element under 35 U.S.C. § 112 ¶6:

> Function: "to communicate one or more document viewers and associated user interface components to the proxy"
>
> Structure: Insufficient structure; indefinite. Ex-1021.

80. PO has proposed that this limitation should be given a plain and ordinary meaning. Ex-1022. It is my opinion that there is no meaningful difference between these constructions as applied to the prior art references at issue in my declaration. I have applied this limitation to include software that operates in the same way as "workflow manager 216," which "manages communication between

document viewers 212, user interface components 208 and, through application proxy 204, with renderer 200." Ex-1001, 5:57-59, Figure 2.

E. "wherein the proxy functions to hide details of each component which makes up the application program from the renderer" (Claim 10)

81. I understand from counsel that, in the related litigation, Petitioner proposed that this limitation is indefinite. *See* Ex-1021. PO has proposed that the limitation is not indefinite. *See* Ex-1022. I have applied this limitation to include that the proxy's component functionality "is not visible to the user." Ex-1001, 5:23-25.

F. "wherein the proxy functions to enable each component which makes up the application program to be modified in a manner that is transparent to the renderer" (Claim 11)

82. I understand from counsel that, in the related litigation, Petitioner proposed that this limitation is indefinite. *See* Ex-1021. PO has proposed that the limitation is not indefinite but did not provide a proposed construction. *See* Ex-1022. I further understand from counsel that during a meet-and-confer in the related litigation, PO stated that it interpreted "transparent to the renderer" as requiring that the renderer is not aware of any modification of "each component which makes up the application program."

83. Another possible interpretation of this limitation is that the proxy's component functionality is "transparent to the renderer" such that the renderer can see through the proxy to additional control options that may be available. Ex-1001,

5:12-14. I have applied this limitation in both possible alternatives to address either of the interpretations outlined above.

84. I reserve the right to offer opinions on any claim constructions proposed in this proceeding

VIII. DETAILED EXPLANATION OF THE UNPATENTABILITY GROUNDS

A. Ground 1: Claims 1-5 and 9-13 are obvious over Beranek in view of Pleyer

1. A POSITA would have been motivated to combine Beranek's teachings with Pleyer's teachings and would have had a reasonable expectation of success.

85. A POSITA would have been motivated to combine the teachings of Beranek and Pleyer and had a reasonable expectation of success in doing so because each relates to controlling how a document is presented for display on a browser of a client machine. Ex-1005, Title, Abstract; Ex-1006, Title, Abstract.

86. Beranek teaches "[a] method of controlling presentation on a client of a Web document formatted according to a markup language and supported on a server, the client including a browser and connectable to the server via a computer network." Ex-1005, 15:33-36 (Cl. 1). Beranek's proxy intercepts, reformats, and controls how a Web document appears in the browser, to the user. Ex-1005, 2:54-61, 2:63-3:2. The proxy will "re-format the Web document according to some given protocol or filter property," such as "adjust[ing] or modify[ing] foreground/background colors, overrid[ing] font types and size, control[ing] display

geometry (e.g., by reserving some dedicated display area), chang[ing] URLs or display[ing] other URLs, embed files, add, remove or reconfigure frames, or control text and table formats." *Id.*, 3:5-14. This enables the "proxy to dynamically adjust the 'look and feel' of a given Web document at the client side." *Id.*, 8:39-42. The "reformatted Web document is then passed to the browser for display." Ex-1005, 3:7-8, Abstract. Beranek teaches that, as the Web document is received from the server, "the HTML is parsed to identify the format of the document and the information therein." Ex-1005, 3:2-5.

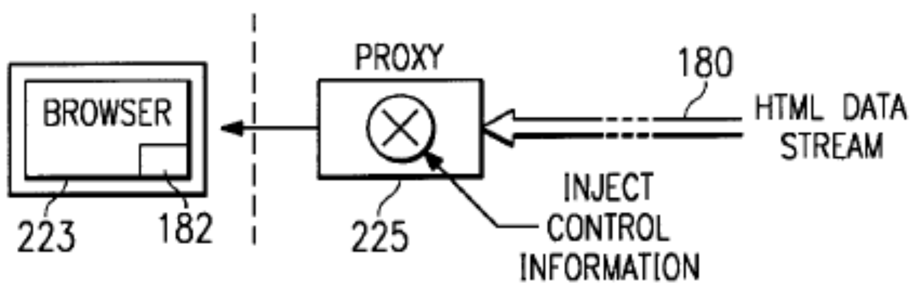


FIG. 9

87. Pleyer teaches a supervisory application 100 that "communicates with hardware devices such as the network interface television tuner, keyboard, and display components though conventional Windows operating system calls." Ex-1006, 5:55-65. The supervisory application communicates the global script 106, which includes browsable content 104 and UI components 105. Pleyer teaches that "controls and dialogs… are implemented as text and graphics documents written in … HTML," and that in order to present options to a user, "global script 106 generates

dialog boxes and controls by specifying local HTML graphical elements." *Id.*, 7:40-45. JavaScript is used to "add interactivity to HTML, to allow for user interaction and feedback, multimedia and animation, and to link HTML to other technologies such as Java, Active X, and plug-ins." *Id.*, 4:12-16. For example, Pleyer teaches that "event handlers implemented by global script 106" includes generating an "on-screen keyboard" and "audio rendering" (*id.*, 9:1-3, 9:29-30), and to "create a new user interface component in response to a user pressing a specific key." *Id.*, 9:49-51.

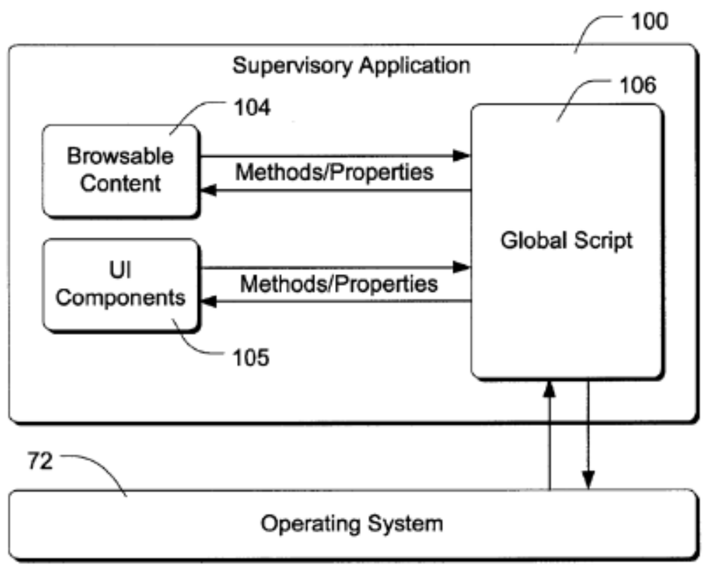


Fig. 3

88. A POSITA would have been motivated to communicate Beranek's web document from the data stream to the proxy with document viewer(s) and associated user interface components, as taught by Pleyer. A POSITA would have understood that Beranek's proxy receives a web document from a workflow manager, which includes a script that implements document viewers and associated user interface

components to display the requested Web document. A POSITA would have further understood that Pleyer provides an example of this well-known approach. Specifically, Pleyer teaches user interface controls and dialogs are "implemented as HTML files or resources" with the overall "logic and sequence of the user interface" being "established by a global script file written in a scripting language such as JavaScript." Ex-1006, 7:18-23. The global script file comes from the supervisory application, a workflow manager.

89. A POSITA would have been motivated for Beranek's workflow manager to communicate one or more document viewers with associated user interface components, implemented by a script, as taught by Pleyer, to control the interaction of displayed controls with underlying hardware and display appropriate graphic elements. Ex-1006, 10:35-40. Specifically, Pleyer teaches that its supervisory application program (workflow manager) runs under an operating system and implements a global script (Ex-1006, 2:11-18), which allows "a software manufacturer [to] sell a single product… to many different manufacturers" (*id.*, 2:42-43) and provides a "very easy" way for a "customer to create an entirely new user interface, and to adapt it to new hardware, without distributing more critical underlying aspects of the software product." *Id.*, 2:49-54. Moreover, Pleyer teaches that its supervisory application program's global script communicates with "HTML components" to produce a full and flexible user interface. *Id.*, 2:38-41. A POSITA

would have been motivated to implement Pleyer's teachings into Beranek to achieve these additional benefits taught by Pleyer.

90. In particular, the Beranek-Pleyer combination would provide a consistent user interface (UI) while browsing the Web. Specifically, Beranek teaches using an HTTP caching proxy to filter and control the display of web pages before the browser shows them, expressly to "enhance the 'look and feel' … and … tailor[] the user interface to the Web appliance environment." Ex-1005, 12:19-23. Pleyer teaches a script-based UI for a TV/Web device in which a global JavaScript plus an extended document object model (DOM) creates dialogs/controls in separate browser instances, precisely to let OEMs deliver a customizable, TV class user interface. A POSITA would have recognized that Beranek's pre-render filtering solves page layout unpredictability, while Pleyer's scripting solves runtime UI behavior and look/feel.

91. The Beranek-Pleyer combination would yield predictable results because both are directed to the same problem and solution: reserving UI space and normalizing third party pages. Beranek explicitly teaches reserving a region of the browser window by injecting control information so that the page "is re-flowed around the reserved area" (e.g., for status/ads/overlays). Ex-1005, 13:41-45. Peyer's global script opens dialogs (modal/modeless) as HTML controls via extended DOM methods, e.g., creating a Toolbar.htm modeless dialog, which needs predictable

screen real estate. Ex-1006, 9:45-10:40; see also id., 10:56-11:5 ("flexible way of implementing a user interface"), Combining them lets the proxy always clear space for Peyer's toolbar/dialogs regardless of the site being visited, a classic "one does the conditioning, the other supplies the controls" pairing.

92. As I described above in section V.A, POSITAs were concerned with consistent presentation of Web documents. Both Beranek and Pleyer acknowledged variability across hardware and pages. Pleyer identifies the lack of standardization and need to adapt to different OEM hardware/UI styles, and proposes a browser plus script architecture so OEMs can customize without recompiling C/C++ code. *See, e.g.*, Ex-1006, 10:55-65. Beranek aims to mask differences between display systems/browsers/OSes by intercepting and rewriting HTML to fit the target device. *See, e.g.*, EX-1005, 14:4-6. A POSITA would naturally pair Pleyer's UI layer with Beranek's page normalization layer to meet both constraints simultaneously.

93. Both further discuss improving readability and layout on TV class displays. Beranek gives concrete filters like removing frames and changing fonts/colors to make pages readable on a "television class display … at a distance." *See* Ex-1005, 10:46-56. Peyer shows a TV based browser device whose UI is built from HTML dialogs and JavaScript event handlers. *See* Ex-1006, Fig. 1, 4:44-56. A POSITA would have thus been motivated to combine Beranek's and Pleyer's

teachings in order to pre-clean the page (as taught in Beranek) so Pleyer's dialogs and controls remain legible and unobtrusive on a TV.

94. A POSITA would have had a reasonable expectation of success in combining the teachings of Beranek and Pleyer. Indeed, as shown below in Modified Figure 9' below, Peyer's supervisory application, which includes its global script with associated browsable content and UI components would have been implemented in conjunction with Beranek's architecture to carry out Beranek's functions (the implementation details of which are left to a POSITA): receiving the content to be passed to the renderer in the appropriate format. A POSITA would be motivated to implement Pleyer's teachings with Beranek to "provide[] a very flexible way of implementing a user interface, which allows individual hardware manufacturers to customize the user interface in an infinite number of ways." Ex-1006, 10:58-60.

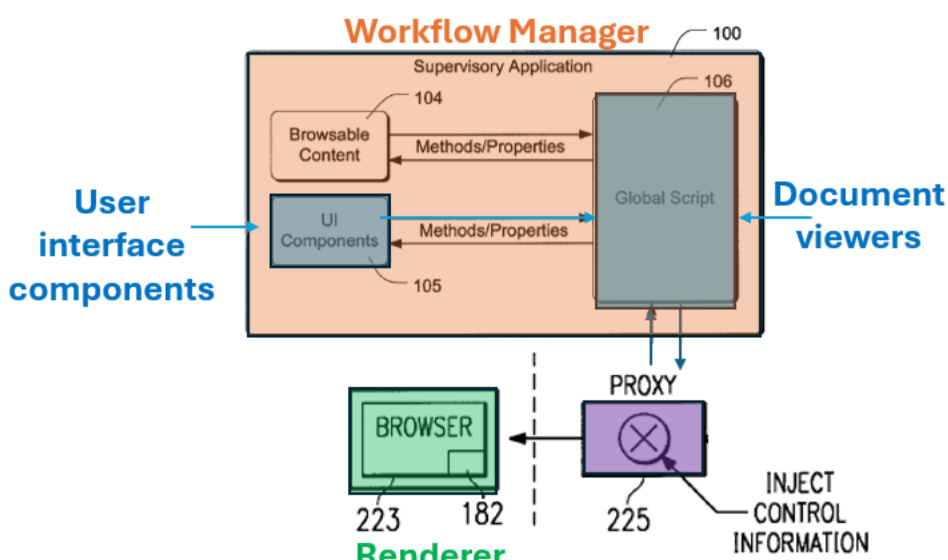


FIG. 9'

95. Moreover, both references describe methods for presenting a requested web document to a user in an appropriate format on a graphical user interface by the user using similar hardware and software components. Ex-1005, Title, 15:33-36 (Cl. 1); Ex-1006, Title, Abstract. The combination would have a reasonable expectation of success due to the predictable synergy without redesigning sites or the UI. Beranek's proxy intercepts HTTP GETs and transparently returns modified pages; the browser "is not 'aware' that the GET request has been serviced locally." Pleyer's supervisory app simply drives browser instances and scripts; it doesn't depend on any specific page authoring. Because each operates independently (network layer vs. script/UI layer), using both is a common-sense way to get consistent overlays/toolbars on every site without touching the sites or rewriting the UI.

96. A POSITA would thus have understood that these references disclose interrelated teachings based on well-understood technologies that would have been amenable to various well-understood and predictable combinations. As I described above, both Beranek and Pleyer use common technology stack and interfaces. Both references rely on standard HTML/JavaScript and mainstream browser components. Peyer's extended DOM exposes methods like Window.external.* to create and manage HTML dialogs; these are ordinary script calls executed by the browser. Beranek's proxy outputs ordinary HTML back to the same browser after rewriting

tags/attributes (e.g., colors, frames), so the page remains standard content that Peyer's UI can coexist with.

97. Additionally, both references implement well defined, published data flow with no incompatibilities. Beranek details the exact flow, intercept, parse, filter, return modified HTML, and even shows the decision points (e.g., activate Dynamic HTML at step 235; parse at 237; filter at 239; return at 241). A POSITA would see that this simply precedes browser rendering. Pleyer details how the supervisory app and global script respond to events and open dialogs in another browser instance, again, a layer above rendering. The layers don't collide; they compose.

98. Both references include explicit teachings about TV/Web appliances and layout control. *See* Ex-1005, 10:46-56; Ex-1006, Fig. 1, 4:44-56. Beranek is written for exactly this use case (web appliance/TV), including space reservation and frame removal on incoming pages. Pleyer is likewise a television based system whose UI is made from HTML components orchestrated by scripts. Beranek guarantees a reserved toolbar strip and consistent layout on every navigated page would predictably allow Pleyer's modeless toolbar/dialogs to render reliably—exactly as intended. This is thus a routine implementation. Pleyer does not restrict or preclude use of a proxy; it only requires a browser instance that renders HTML and runs scripts, precisely what Beranek feeds. Beranek, in turn, is agnostic to application level scripts sitting atop the browser; it just filters HTML sources and

returns them to the browser. Given the ubiquity of HTTP proxies and browser scripting at the time, a POSITA would expect the pieces to just work.

2. Independent Claims 1 and 9

a. Element 1[pre]: A computer implemented system for providing a graphical user interface, comprising:

Element 1[A]: a non-transitory computer readable medium;

Element 9[pre]: A method for providing a graphical user interface, comprising:

99. Beranek-Pleyer teaches the preambles of claims 1 and 9, to the extent that they are limiting.

100. Beranek teaches "[a] method of controlling presentation on a client of a Web document formatted according to a markup language and supported on a server, the client including a browser and connectable to the server via a computer network." Ex-1005, 15:33-36 (Cl. 1). Beranek also teaches "[a] ***computer program product in computer-readable media*** for use in controlling presentation on a client of a Web document formatted according to a markup language (HTML) and supported on a server, the client including a browser and connectable to the server via a computer network." Ex-1005, 16:43-47 (Cl. 19).

101. Beranek teaches that the browser "sits on a given machine (e.g., a personal computer having its ***own display system***…"). Ex-1005, 13:2-6. As shown in Figure 9 below, browser 232 is available on a display screen, such as a graphical

user interface. Moreover, Beranek teaches a "client computer, for example, a Web appliance, [] connectable to the Internet and comprises a processor having an operating system, a browser application run by the operating system, a cache, and a proxy." Ex-1005, 2:54-58; *see also id.,* 4:58-5:17 ("client machine is a personal computer such as a desktop of notebook computer"). A POSITA would have understood that controlling how a Web document is rendered in a browser necessarily provides and uses a GUI—the browser window, menus, buttons, and controls comprise the graphical user interface through which the user interacts. A POSITA would have understood that Beranek's method for controlling presentation on a client computer is a method for providing a graphical user interface because the client computer would have a graphical user interface to display the Web document that Beranek controls and displays.

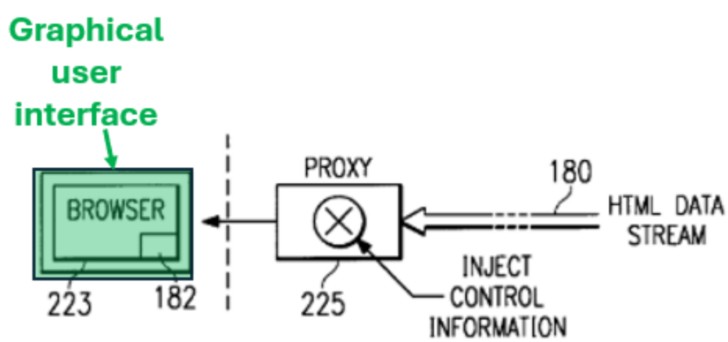


FIG. 9

b. Elements 1[B]/9[A]: [operating] a renderer defined to render a graphical user interface (GUI) for an application program;

102. Beranek-Pleyer teaches Elements 1[B]/9[A].

103. Beranek teaches operating a renderer defined to render a GUI for an application program. As discussed in Section VII, I have applied this limitation as including software that renders a graphical user interface for an application program that same way as the patent, which would include software such as "a DHTML browser." Ex-1001, 4:53, 5:23-24, 7:19.

104. Beranek's method operates a renderer to render a GUI for the application program by teaching that its method intercepts, reformats, and controls how a Web document ***appears*** in the browser to the user. Ex-1005, 2:54-61. Specifically, Beranek teaches that the "reformatted Web document is then passed to the ***browser for display***." Ex-1005, 3:7-8, Abstract. Beranek further explains that an "object of this invention is to implement a 'dynamic HTML' [DHTML] function in a standalone Web appliance or system to thereby modify the appearance of a web document prior to its display on the browser." *Id*., 2:36-39. A POSITA would have understood Beranek's software instructs the browser application to render a GUI for the application program using its "dynamic HTML" (DHTML) functioning browser. This is consistent with the description in the '428 Patent, stating that its "renderer 200 is a DHTML browser." Ex-1001, 5:23-34.

c. Elements 1[C]/9[B]: [(i) [operating] a proxy defined to provide each component necessary for rendering the GUI to the renderer, [(ii)] wherein the proxy appears to the renderer as a single monolithic application program, and [(iii)] wherein the proxy is defined to determine changes required to be made to the GUI

for the application program during execution of the application program, [(iv)] wherein a given change determined by the [proxy]/[ox] is one of rendering a new layout in the GUI and updating a present layout in the GUI; and

105. Beranek-Pleyer teaches Elements 1[C]/9[B].

106. Beranek-Pleyer teaches operating a proxy defined to provide each component necessary for rendering the GUI to the renderer. Specifically, Beranek teaches a "client-side caching proxy," (purple below) that "intercept[s] the Web document and then dynamically rewrite[s] the document before it is displayed on the browser." Ex-1005, 2:63-3:2. This enables the "proxy to dynamically adjust the 'look and feel' of a given Web document at the client side of the connection." *Id.*, 8:39-42. The proxy "is implemented in software and includes a cache 227." *Id.*, 8:24-26. Specifically, as shown in Figure 9, proxy 225 (purple), intercepts the data stream and "inject[s] new control information … in order to affect how the Web content is ultimately displayed." *Id.*, 13:9-14.

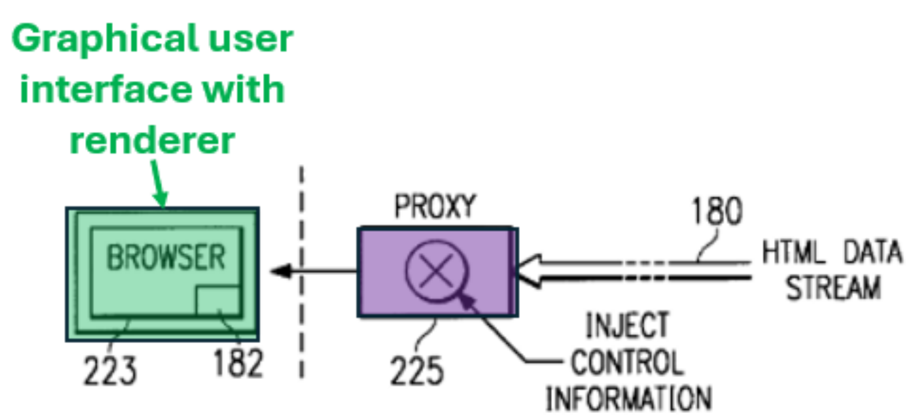


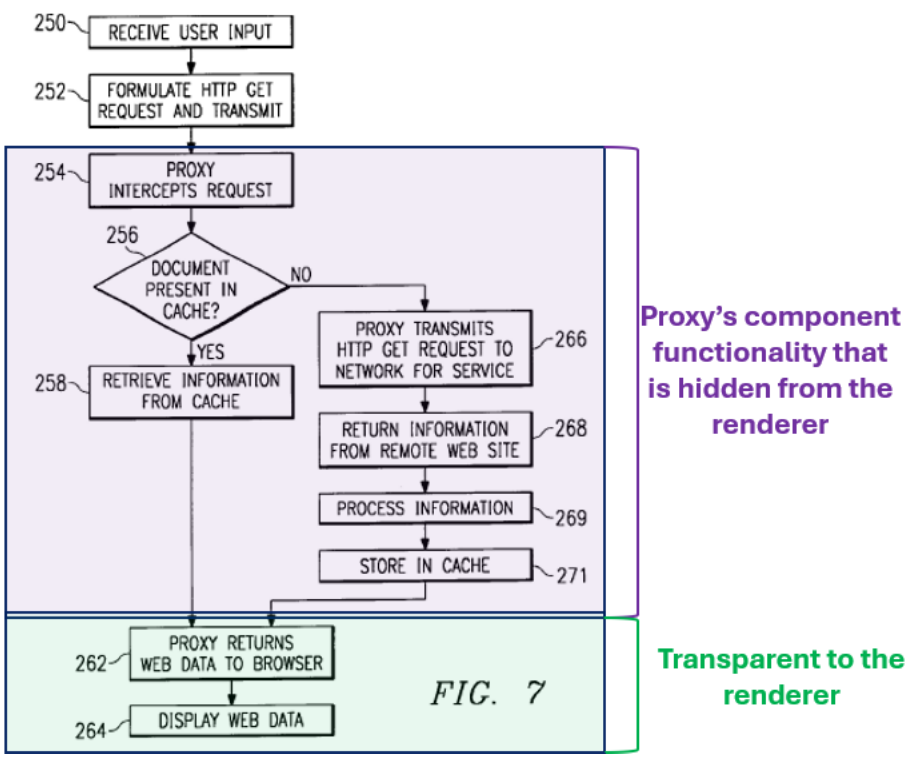
FIG. 9

107. In Beranek, the proxy appears to the renderer as a "single monolithic application program." As discussed in Section VII, I have applied this limitation to include software that operates in the same way as "application proxy 204," which is "a single, monolithic application program and all details of the components which make up the application program are essentially hidden from renderer 200 which need only interface to application proxy 204." Ex-1001, 5:8-12.

108. As shown above in Figure 9, proxy 225 appears as a "single, monolithic application program and all details of the components which make up the application program are essentially hidden" from the renderer because the renderer only receives the "reformatted Web document" from proxy 225. Beranek's proxy 225 is "a piece of standalone or downloadable code that intercepts and filters the Web document retrieved over the computer network." Ex-1005, 12:14-16. Proxy 225's functions of filtering, managing/processing, and monitoring the network-retrieved data and controlling the "look and feel" of the components that make up the application program on the browser interface are hidden from the renderer by being located in or behind proxy 225. Ex-1005, 8:63-67.

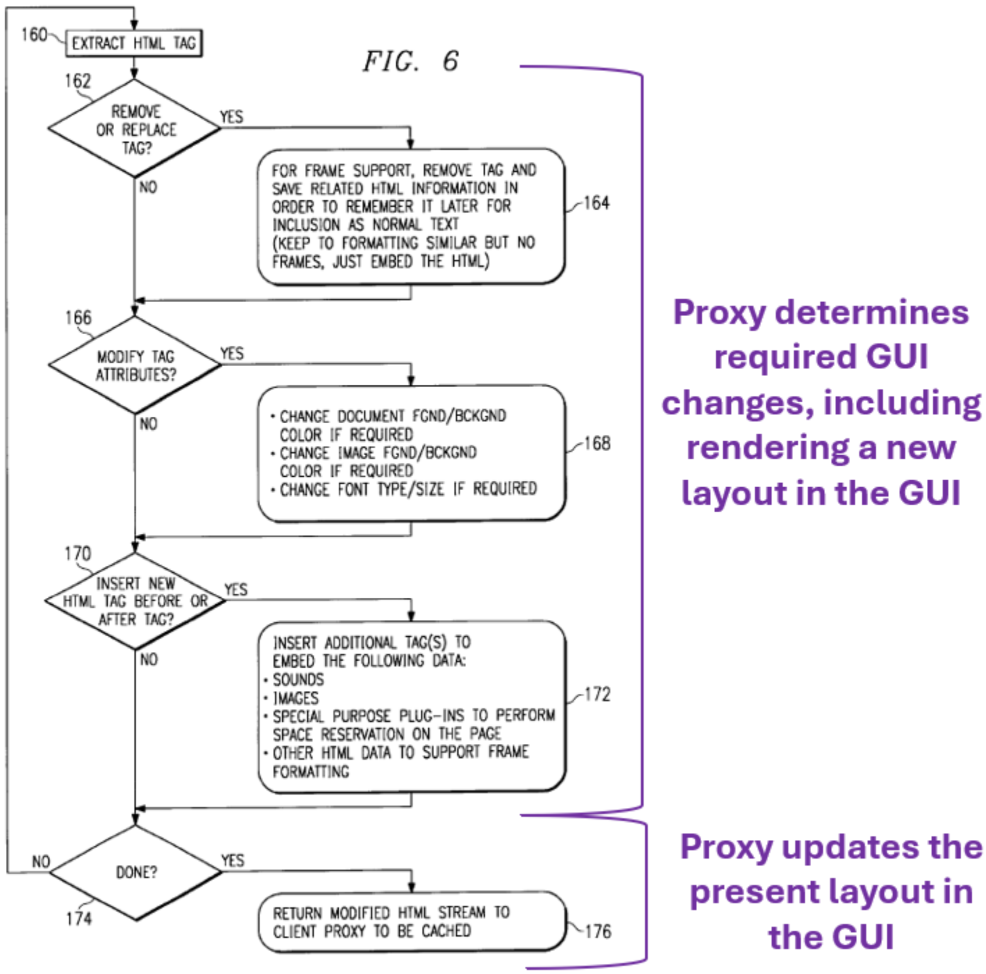
109. For example, as shown below in Beranek's Figure 7, the proxy intercepts a request from the browser (i.e., the claimed renderer) at step 254 and performs the proxy's functions (steps 256, 258, 266, 268, 269, and 271, purple) before returning the web data to the browser; "[t]he ***browser is not 'aware'*** that the

GET request has been serviced locally." *Id.*, 11:33-50. Therefore, the proxy implements "'dynamic HTML' processing or 'filtering' of the Web document," but the proxy does not provide those details to the renderer to render the Web document on the browser at step 264. Ex-1005, 9:5-7 ("This operation may be effected without the browser being aware that the service is being handled locally.").



110. Beranek teaches that the proxy is defined to determine changes required to be made to the GUI for the application program during execution of the application program, where the given change is rendering a new layout in the GUI and updating a present layout in the GUI. Beranek's proxy includes a filter mechanism to "re-format a retrieved HTML document before the document is presented on the browser." Ex-1005, 9:8-10. The proxy is used to modify and/or

process data before it is displayed on the user's interface, such that the "data font or presentation to be modified, supplemented or otherwise adapted to the particular display environment." *Id.*, 12:17-19. Specifically, as shown below in Figure 6, the proxy determines the HTML tags from the Web document being parsed and to remove, replace, or modify tag attributes (162, 166). *Id.*, 10:42-11:3. Beranek's proxy determines a change in rendering and updating the UI with a new layout when determining to remove the tag and save the HTML information for inclusion as normal text to support the frame in the client's display (*id.*, 10:58-65) and/or determines whether to "change document foreground or background color…, the color of a particular image…, or the font type or size of certain text." *Id.*, 11:4-7. Beranek's proxy thus provides a tool to enhance and tailor the "look and feel" of the layout of the data displayed and update the "user interface to be tailored to the Web appliance environment." Ex-1005, 12:19-23.



d. Elements 1[D]/9[C]: [operating] a workflow manager defined to communicate one or more document viewers and associated user interface components to the proxy for provision to the renderer as components necessary for rendering the GUI[,]/[.]

111. Beranek-Pleyer teaches elements 1[D]/9[C].

112. Beranek-Pleyer teaches operating a workflow manager defined to communicate one or more document viewers and associated user interface components to the proxy for provision to the renderer as components for rendering the GUI. As discussed in Section VII, I have applied this limitation to include

software that operates in the same way as "workflow manager 216," which "manages communication between document viewers 212, user interface components 208 and, through application proxy 204, with renderer 200." Ex-1001, 5:57-59, Figure 2.

113. As shown in Figure 9, Beranek teaches that its proxy is given the Web document to filter and format by a data stream 180, received from the computer network. Ex-1005, 13:6-16.

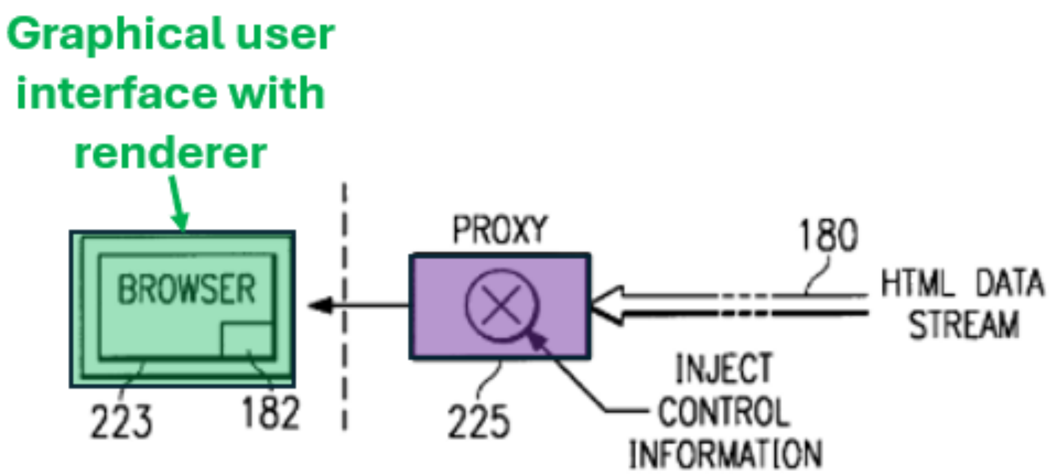


FIG. 9

114. Similarly in Figure 10 (below), Beranek teaches that in step 184, the data stream is received from "a server at which the document is supported" and in step 186, the stream is intercepted and parsed prior to its delivery to the browser. Ex-1005, 13:24-26.

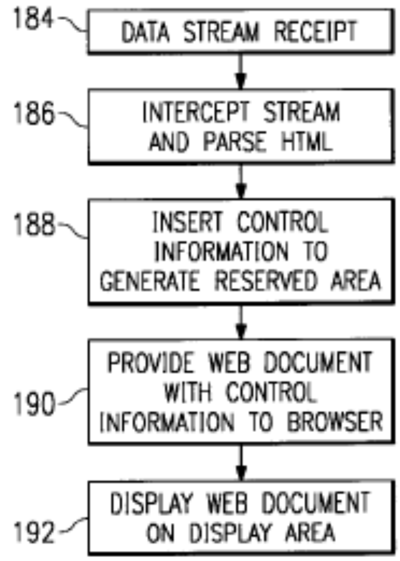


FIG. 10

115. A POSITA would have been motivated to implement a workflow manager with Beranek's method, as taught by Pleyer, to manage communications of the document viewers and user interface components to be rendered to the proxy. As shown below in Figure 3, Pleyer teaches a supervisory application 100 that "communicates with hardware devices such as the network interface television tuner, keyboard, and display components though conventional Windows operating system calls." Ex-1006, 5:55-65. The supervisory application communicates the global script 106, which includes browsable content 104 and UI components 105. That is, a POSITA would have been motivated to pair Pleyer's workflow orchestration (supervisory application + global script + dialogs/viewers) with Beranek's proxy pipeline to ensure that the chosen viewer/components are consistently delivered and rendered at the client, including dynamic adjustments of

look-and-feel via the proxy, exactly the problem Beranek solves. This combination uses standard HTML/JavaScript/browser instances and a client-side proxy; the technologies are expressly compatible, giving a reasonable expectation of success. *See* Section V.

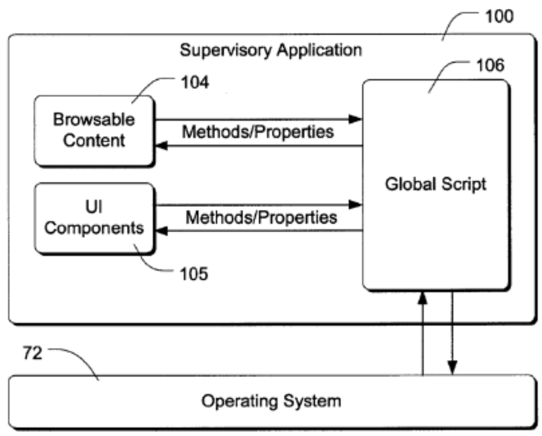


Fig. 3

116. The Beranek-Pleyer combined system would thus take the form depicted below, in Figure 9':

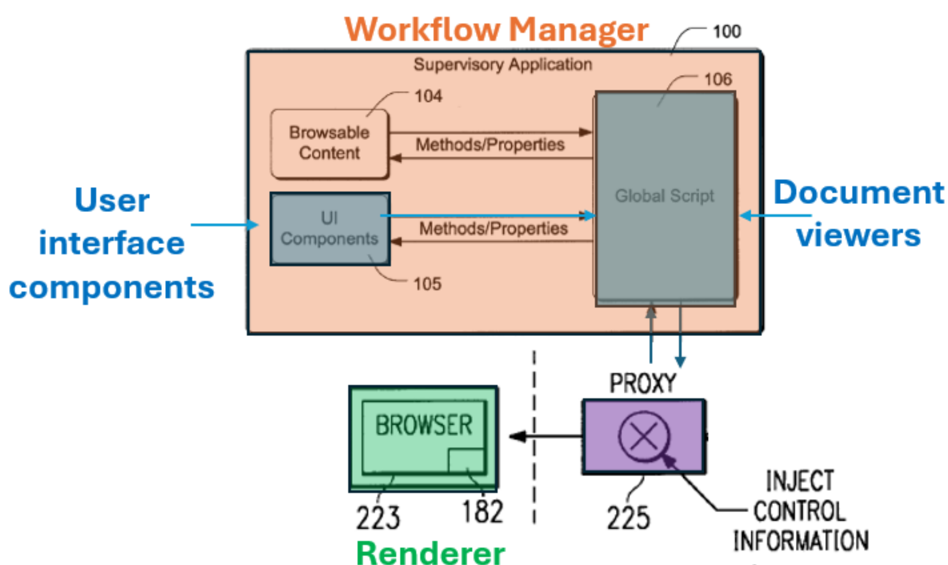


FIG. 9'

117. Pleyer's workflow manager (supervisory application 100) communicates one or more document viewers implemented as a global script. Pleyer teaches that "visual components of a user interface [are] implemented through a combination of documents or source files written in conventional internet hypertext formats." Ex-1006, 1:60-64. For example, hypermedia content, such as "text, images, sounds, and actions" are provided to a user as "individual 'documents' or 'pages' with embedded control information." *Id.*, 3:1-5, 3:10-13. Pleyer teaches that "controls and dialogs… are implemented as text and graphics documents written in … HTML," and that in order to present options to a user, "***global script 106*** generates dialog boxes and controls by specifying local HTML graphical elements." The global script generates user interface components using the "ShowModalDialog(), ShowModelessDialog( ), CloseModalDialog( ), and CloseModelessDialog( ) methods, referred to herein as 'open' and 'close' interface methods." *Id.*, 8:56-60. The global script is the document viewer that the workflow manager communicates to the proxy to render the document for the user's observation and interaction with data appropriate to the application program, in all its possible contexts.

118. Pleyer teaches that the document viewer includes associated user interface elements. Specifically, Pleyer teaches that "appropriate HTML elements" are displayed to the user for the user to interact with device hardware. Ex-1006, 2:4-

10. Pleyer teaches that "executable programs, text or multimedia documents, sound clips, audio segments, still images, computers, directors, and other hyperlinks" may be rendered in the displayed document. *Id*., 3:47-55. The formatting of the document includes "HTML markup codes or 'tags,'" which instruct the Web browser "how to render and print documents." *Id*., 4:1-5. JavaScript is used to "add interactivity to HTML, to allow for user interaction and feedback, multimedia and animation, and to link HTML to other technologies such as Java, Active X, and plug-ins." *Id*., 4:12-16. For example, Pleyer teaches that "event handlers implemented by global script 106" includes generating an "on-screen keyboard" and "audio rendering" (*id*., 9:1-3, 9:29-30), and "create a new user interface component in response to a user pressing a specific key." *Id*., 9:49-51. Pleyer further confirms that the document viewer includes associated UI elements, with the global script and dialogs handling user events and even integrating device interactions, again the workflow manager delivering both the viewer and its UI components for rendering.

119. A POSITA would have been motivated to implement Beranek-Pleyer together so that the workflow manager communicates one or more document viewers with associated user interface components, implemented by a script, as taught by Pleyer, to control the interaction of displayed controls with underlying hardware and display appropriate graphic elements and for the additional reasons provided in Section VIII.A.1. Ex-1006, 10:35-40. Besides the functional incentives

(consistent client rendering, dynamic control, standard HTML/JS), a POSITA would have understood that GUI components were commonly packaged and supplied to renderers: e.g., constructing a GUI by inserting visual controls and dynamically assembling the GUI from available modules was routine, predictable engineering. *See, e.g.*, Section V.

e. Element 1[E]: wherein the renderer, the proxy, and the workflow manager are executed from the non-transitory computer readable medium.

120. Beranek-Pleyer teaches element 1[E].

121. Beranek teaches that its software components, e.g. the renderer, proxy, and workflow manager, are executed from the non-transitory computer readable medium. Beranek teaches "[a] computer program product in computer-readable media for use in controlling presentation on a client of a Web document formatted according to a markup language (HTML) and supported on a server, the client including a browser and connectable to the server via a computer network." Ex-1005, 16:43-47 (Cl. 19). Beranek also teaches that the "preferred implementations of the client side or server side proxy… is as a set of instructions (program code)." *Id.*, 14:63-66. The instructions may be "stored in another computer memory, for example, in a hard disk drive, or in a removable memory such as an optical disk (for eventual use in a CD ROM) or floppy disk (for eventual use in a floppy disk drive), or downloaded via the Internet or other computer network." *Id.*, 14:66-15:5. Beranek

also teaches that "one of ordinary skill in the art would also recognize such method may be carried out in hardware, in firmware, or in more specialized apparatus constructed to perform the required method steps." *Id.*, 15:8-10.

3. Dependent Claims 2-5 and 10-13

a. Claim 2: A computer implemented system for providing a graphical user interface as recited in claim 1, wherein the proxy enables details of each component which makes up the application program to be hidden from the renderer.

Claim 10: A method for providing a graphical user interface as recited in claim 9, wherein the proxy functions to hide details of each component which makes up the application program from the renderer.

122. Beranek-Pleyer teaches Claims 2 and 10.

123. Beranek-Pleyer teaches that the proxy hides details of each component which makes up the application program from the renderer. As discussed in Section VII, I have applied this limitation to include that the proxy's component functionality "is not visible to the user." Ex-1001, 5:23-25.

124. Beranek teaches that its proxy hides details of each component which makes up the application program from the renderer as explained in Section VIII.A.2.c (elements 1[c]/9[b]). Specifically, Beranek's proxy 225 appears just as "application proxy 204," which is "a single, monolithic application program and all details of the components which make up the application program are essentially hidden from renderer 200 which need only interface to application proxy 204." Ex-

1001, 5:8-12. For example, Beranek's proxy 225's components are hidden from the renderer because the renderer "***is not 'aware'***" how its request was serviced by the proxy. Ex-1005, 11:33-50. The renderer receives the "reformatted Web document" from the proxy after it is reformatted by the proxy's components without the details of how the Web document was reformatted. Therefore, while the proxy implements "'dynamic HTML' processing or 'filtering' of the Web document," the proxy does not provide those details to the renderer to render the Web document on the browser. Ex-1005, 9:5-7 ("This operation may be effected ***without the browser being aware*** that the service is being handled locally.").

125. The renderer simply receives a reformatted document and renders it. The renderer is decoupled from multiple cooperating UI and hardware-interaction components (e.g., dialog generators, event handlers, device-control routines) and interacts only with the rendered result (HTML/dialogs); it does not learn which components were used or how they were composed. This provides a proxy that enables/functions to hide the details of each component making up the application program from the renderer while still delivering render-ready UI content.

b. Claim 3: A computer implemented system for providing a graphical user interface as recited in claim 1, wherein the proxy enables each component which makes up the application program to be modified in a manner that is transparent to the renderer.

Claim 11: A method for providing a graphical user interface as recited in claim 9, wherein the proxy functions to enable each component which makes up the application program to be modified in a manner that is transparent to the renderer.

126. Beranek-Pleyer teaches Claims 3 and 11.

127. Beranek-Pleyer teaches that the proxy enables each component which makes up the application program to be modified in a manner transparent to the renderer. As discussed in Section VII, I understand from counsel that PO stated during a meet-and-confer that it interpreted "transparent to the renderer" as requiring that the renderer is not aware of any modification of "each component which makes up the application program." For completeness, I have addressed PO's proposed alternative construction here. Petitioner's proposed alternative construction is presented in Ground 3.

128. Under PO's interpretation, Beranek-Pleyer teaches Claims 3 and 11. Beranek teaches that its proxy enables components of the application program to be modified in a manner that is transparent to the renderer while the renderer is not aware of any modification. As described in Section VIII.A.2.c (elements 1[c]/9[b]), Beranek's proxy 225 hides the components of the application program from the

renderer because the renderer "***is not 'aware'***" how its request was serviced by the proxy. Ex-1005, 11:33-50; *see also id*., 9:1-7. The renderer receives the "reformatted Web document" (i.e., components of the which makes up the application program) from the proxy, after it is reformatted by the proxy without the details of how the Web document was reformatted, so that Beranek's proxy 225 appears as "a single, monolithic application program and all details of the components which make up the application program are essentially hidden." *See supra* section VIII.A.2.c (elements 1[c]/9[b]). Because the renderer "is not 'aware'" how its request is serviced, the renderer is able to modify the application program's components without being aware of the proxy's reformatting the Web document/application program. Beranek describes concrete, granular modification operations the proxy performs before rendering: removing/replacing FRAME tags and saving frame content for re insertion, changing attributes (e.g., background color, font size), inserting new tags (e.g., to embed plug ins, sounds, images), and then returning the modified HTML stream to the browser. These edits are applied prior to display, so the renderer sees only the final result and remains unaware that any component was changed. In the Beranek-Pleyer combination, Pleyer's components rendered are HTML/markup resources (dialogs, controls, ActiveX backed UI) chosen and created by the global script/supervisory app. Because Pleyer's components are markup specified components, any of them can be rewritten, substituted, or re parameterized by the proxy (as taught in Beranek) without changing the renderer. That is, each component is modifiable at the proxy layer while remaining transparent to the renderer.

129. For example, Beranek's proxy enables components of the application program to be modified while being invisible to the renderer. The '428 Patent explains that the proxy enables the "components of the application program to be modified in a manner which is transparent to the renderer" and provides an example of "an improved 'Edit Picture' document viewer… substitut[ed] in the application program." Ex-1001, 5:12-17. Just as the improved "Edit Picture" document viewer is rendered without the renderer being aware of how the proxy handled the modification, Beranek teaches that its proxy updates the Web document without making the details of the updating known to the renderer. Specifically, Beranek's DHTML browser allows for the renderer to display dynamic HTML functions, allowing a web page's content, style, and structure to be edited after the web page has been loaded, without requiring a full page reload.

130. As shown in Figure 5, when the proxy determines that the DHTML function is active in step 235, the "proxy identif[ies] the HTML format and other information therein that may need to be ***supplemented, modified or otherwise changed*** to affect the appearance of the document" in step 237. Ex-1005, 10:13-20. The proxy filters the parsed information "according to a given filter property," "***defined by the system" or "defined by the user using the remote control***." *Id.*,

10:20-24. The document is modified and passed to the renderer according to the filtering defined by the system or the user. *Id.*, 10:25-27. Therefore, the proxy enables the components of the application program (the rendered Web document) to be edited in a manner that is transparent to the renderer by following filtering rules defined by the renderer, while hiding the details of how the edit is performed from the renderer. Beranek's proxy remains invisible to the renderer but allows for components of the application program to be modified in a manner transparent to the renderer. That is, Beranek's flow explicitly shows step 235/237 logic: if the dynamic HTML function is active, the proxy parses the document and identifies HTML that "may need to be supplemented, modified or otherwise changed," then applies filter properties and passes the (now modified) document to the browser. This is modification performed by the proxy and opaque to the renderer. Those filter properties can be system defined or user defined, and Beranek further claims re-formatting "in accordance with at least one filter property defined by a user,"confirming that proxy side modifications can be driven dynamically while still shielded from the renderer. Because Beranek-Pleyer's components (including those with scripts) are markup based resources selected and displayed by a second browser instance, the proxy's rewriting primitives (tag/attribute change, insertion, and even script modification) apply uniformly to each component, preserving renderer transparency across the entire component set. In summary, "each component …

modified … transparent to the renderer" is satisfied by (i) Pleyer's componentized, browser rendered UI (second browser instance + markup based components + script/dialog management), and (ii) Beranek's proxy that rewrites any markup/script content and returns only the finished result to the browser that is "not 'aware'" of the proxy's modifications.

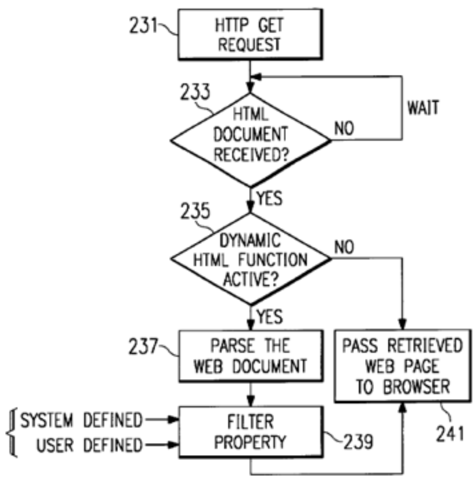


FIG. 5

c. **Claim 4: A computer implemented system for providing a graphical user interface as recited in claim 1, wherein each of the one or more document viewers includes functions to enable the renderer to display a current document in an appropriate format.**

Claim 12: A method for providing a graphical user interface as recited in claim 9, wherein each of the one or more document viewers includes functions to enable the renderer to display a current document in an appropriate format.

131. Beranek-Pleyer teaches Claims 4 and 12.

132. Beranek-Pleyer teaches that its document viewers include functions to enable the renderer to display a current document in an appropriate format. Beranek teaches one or more document viewers, which include functions to enable the renderer to display a current document in an appropriate format by identifying the format of the Web document as it is received from the server (Ex-1005, 3:2-5) and "***re-format*** the Web document according to some given protocol or filter property." *Id*., 3:5-8. For example, Beranek teaches "adjust[ing] or modify[ing] foreground/background colors, overrid[ing] font types and size, control[ing] display geometry (e.g., by reserving some dedicated display area), chang[ing] URLs or display[ing] other URLs, embed files, add, remove or reconfigure frames, or control text and table formats." *Id*., 3:5-14. Beranek thus includes functions to enable the renderer to display the document in an appropriate format after the proxy has reformatted the received Web document from the data stream.

133. A POSITA would have been motivated to implement Beranek's document viewers to include functions to enable the renderer to display a current document in an appropriate format, as detailed by Pleyer. Pleyer teaches that "visual components of a user interface [are] implemented through a combination of documents or source files written in conventional internet hypertext formats." Ex-1006, 1:60-64. For example, hypermedia content, such as "text, images, sounds, and actions" are provided to a user as "individual 'documents' or 'pages' with embedded

control information." *Id*., 3:1-5, 3:10-13. Pleyer teaches that "controls and dialogs… are implemented as text and graphics documents written in … HTML," and that in order to present options to a user, "***global script 106*** generates dialog boxes and controls by specifying local HTML graphical elements." *Id*., 7:40-45. The global script generates user interface components using the "ShowModalDialog(), ShowModelessDialog( ), CloseModalDialog( ), and CloseModelessDialog( ) methods, referred to herein as 'open' and 'close' interface methods." *Id*., 8:56-60. Pleyer's global script includes functions for the renderer to display a current document in an appropriate format because it communicates what to render for the user's observation and interaction with data appropriate to the application program, in all its possible contexts.

d. Claim 5: A computer implemented system for providing a graphical user interface as recited in claim 1, wherein each document viewer includes a respective user interface requirements specification referencing one or more user interface components associated with the document viewer.

Claim 13: A method for providing a graphical user interface as recited in claim 9, wherein each document viewer includes a respective user interface requirements specification referencing one or more user interface components associated with the document viewer.

134. Beranek-Pleyer teaches Claims 5 and 13.

135. Beranek-Pleyer teaches that each document viewer includes a respective user interface requirements specification referencing one or more user interface components associated with the document viewer. Specifically, Pleyer teaches user interface components such as "options" are presented to a user by the global script 106 generating "dialog boxes and controls by specifying local HTML graphical elements… to the supervisory application" which then "passes" such "***specifications*** to a browser…, [to] display[] the specified HTML graphical elements in conjunction with whatever material is already being displayed as a result of user browsing." Ex-1006, 7:41-49. Pleyer teaches specific events and the resulting behavior to be handled by the global script. *Id.*, 9:1-31.

136. For example, in response to a user pressing a specific key, Pleyer implements an "event handler in the global script" to create a new user interface component, by showing a toolbar and dialog window. *Id.*, 9:47-63. In another example, a "TVTuner component" can be implemented as a user interface component rendered by displaying an "ActiveX TV Tuner control as specified by the class identifier (CLASSID)" so that a "TV viewing window in response to a user pressing a specific key" is opened. *Id.*, 9:64-10:15. Pleyer's "event handlers" are the respective user interface requirements specification that reference one or more user interface components associated with the document viewer, the global script, because the event handlers are the respective user interface requirements

specification that are handled by the global script according to the requirements for that event.

B. Ground 2: Claims 3 and 11 are obvious in view of Beranek-Pleyer in view of Carter.

1. A POSITA would have been motivated to combine Beranek-Pleyer's teachings and Carter's and would have had a reasonable expectation of success.

137. A POSITA would have been motivated to combine Beranek-Pleyer's teachings and Carter's teaching and had a reasonable expectation of success in doing so because each relates to controlling how a document is presented for display on a browser of a client machine. Ex-1005, Title, Abstract; Ex-1006, Title, Abstract; Ex-1007, Abstract.

138. As discussed above, Beranek teaches "[a] method of controlling presentation on a client of a Web document formatted according to a markup language and supported on a server, the client including a browser and connectable to the server via a computer network." Ex-1005, 15:33-36 (Cl. 1). Beranek intercepts, reformats, and controls how a Web document appears in the browser, to the user. Ex-1005, 2:54-61. Specifically, a "client-side caching proxy," "intercept[s] the Web document and then dynamically rewrite[s] the document before it is displayed on the browser." Ex-1005, 2:63-3:2. In Beranek-Pleyer's system "[t]he ***browser is not 'aware'***" of how the user's request is serviced by the proxy because the functions are hidden. Ex-1005, 11:33-50.

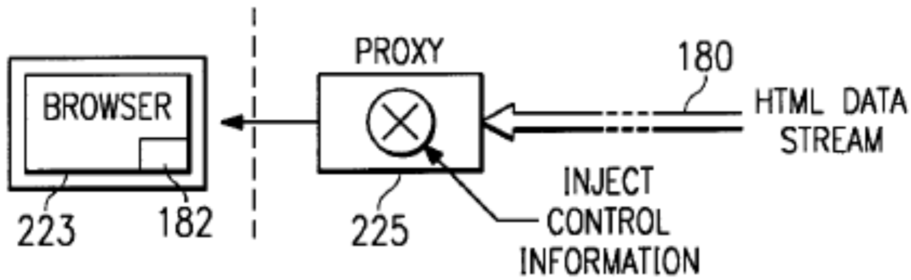


FIG. 9

139. Carter teaches "dynamically construct[ing] a graphical user interface associated with an application" by determining which command classes to "instantiate" and constructing "the graphical user interface for the application such that the graphical user interface contains the additional functions of the successfully instantiated command classes." Ex-1007, Abstract. Carter teaches a constructor 3120, which "locates any packages (or classes in a package)." Ex-1007, 5:27-37. The program will determine "which classes are available and then construct a graphical user interface ***to reflect all of the classes***." *Id.*, 2:56-57. Figure 3 shows a command class library which stores multiple command classes 3100-3400 that include additional application functionality. *Id.*, 4:27-30. Carter's command classes are rendered on the application so that the user may select the additional functionality, which then causes a new layout in the GUI and updating a present layout in the GUI for the user. *See, e.g.*, Figures 5A, 5B.

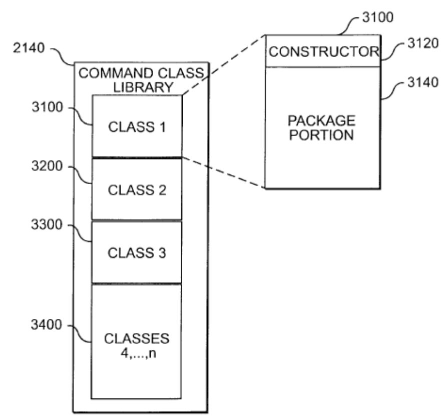


FIG. 3

140. A POSITA would have been motivated for Beranek-Pleyer's proxy functionality to be transparent to the renderer, and thus the user, as taught by Carter. so that they can be invoked, managed, and undone through a consistent interface directly by the user. Carter determines the form of a graphical user interface associated with an application based on a run-time determination of the available features in order to "provide only those additional functions that the customer desires" without "the creation of several versions of the application." Ex-1007, 1:56-58, 2:1-3, 6:15-18. That is, a POSITA working on browser-based UIs would have seen a natural, technical fit between (i) Beranek's proxy that intercepts and rewrites HTML before the browser renders it (to normalize layout and inject UI placeholders), (ii) Pleyer's script driven architecture that opens and manages UI components written in HTML in a second browser instance, and (iii) Carter's run

time mechanism that discovers available feature modules and adds or removes GUI controls accordingly, without rebuilding the program.

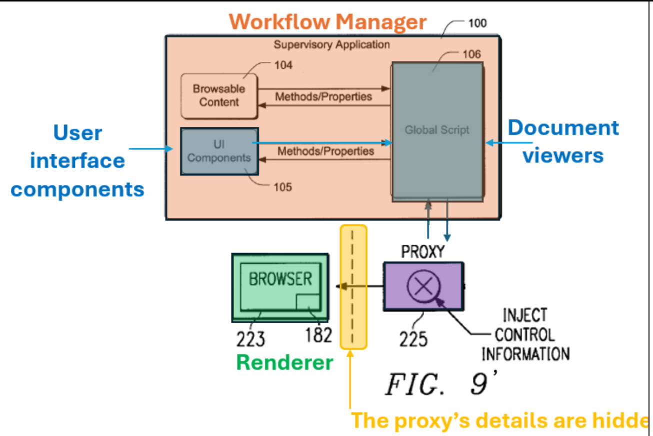
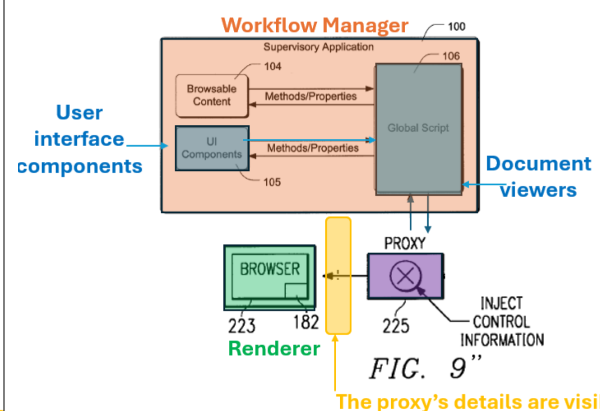
141. In particular, the combination solves a recognized, practical problem in Beranek-Pleyer. As I discussed above, both Beranek and Pleyer discuss the highly variable TV/web devices where different OEMs want unique UIs and hardware integration. That variability naturally creates cross device rendering issues. Beranek addresses precisely that by normalizing/injecting markup at the proxy so pages render predictably "across multiple Web content display system types." A POSITA using Beranek-Pleyers's proxy to inject placeholders/controls and to keep the HTML UI consistent on heterogeneous devices, without touching the browser engine and Carter furthers that goal. Implwmenting Carter's teachings would yield predictable improvement of the same technology stack by enabling features on the fly, without redeployment. Specifically, OEMs and service providers frequently want to ship/add features after the base image. Carter teaches a ready made pattern: at run time, detect which feature modules ("command classes") are present, and insert/remove the corresponding controls in the GUI without updating the program. A POSITA would have been motivated to use Carter's run time gating to decide which HTML dialogs/components Beranek-Pleyer should open, giving modularity and upgradability.

142. Indeed, the three references include natural layering with complementary roles. The three pieces sit at different, non-conflicting layers: Beranek (proxy): preprocess the HTML stream—reserve UI regions, inject tags, and normalize layout before the browser sees it. Peyer (browser & scripts): open/close dialogs and components in a second browser instance under a global script when events fire (e.g., a feature is available). Carter (run time discovery): determine available modules and direct the UI to show the corresponding controls. Using them together is nothing more than "using prior art elements according to their established functions" to meet a common goal (dynamic, device robust browser UI).

143. A POSITA would have been motivated to combine Beranek-Pleyer and Carter's teachings because design incentives point the same way. For example, Beranek-Peyer explicitly targets highly customizable UIs and exposes UI in HTML/JavaScript so OEMs can iterate rapidly with Beranek eliminates the cost of producing multiple page versions for different displays by rewriting one page at the proxy. Carter further removes the need to repackage the program to enable new features. A single system that adapts at run time and renders consistently anywhere would have been an obvious product goal in that era.

144. A POSITA would have had a reasonable expectation of success in combining the teachings of Beranek-Pleyer and Carter. Indeed, as shown below in Modified Figure 9'' below, Carter's command classes (that add functionality) enable

the functions that are performed by Beranek's proxy to be modified in a manner that is transparent to the renderer because the user can see the available options on how to change the layout of the GUI so that the line between the renderer and proxy is removed. *Id.*

Beranek-Pleyer	Beranek-Pleyer-Carter
	

145. Moreover, the references describe methods for presenting a requested web document to a user in an appropriate format on a graphical user interface by the user using similar hardware and software components. Ex-1005, Title, 15:33-36 (Cl. 1); Ex-1006, Title, Abstract; Ex-1007, Abstract. A POSITA would thus have understood that these references disclose interrelated teachings based on well-understood technologies that would have been amenable to various well-understood and predictable combinations. Combining them yields a predictable, incremental system: at run time the system discovers which functions are available (Carter), exposes the corresponding controls as HTML dialogs and components and uses a

proxy to insert/normalize the necessary markup so the UI renders consistently across devices (Beranek-Pleyer). A POSITA would have expected this to work because all three teach standard, widely compatible technologies (HTTP / HTML / JavaScript / Java) and operate at complementary layers (proxy → browser/markup → run time feature gating). I discussed how these were well-known technologies in Section V.

2. Dependent Claims 3 and 11

a. Claim 3: A computer implemented system for providing a graphical user interface as recited in claim 1, wherein the proxy enables each component which makes up the application program to be modified in a manner that is transparent to the renderer.

Claim 11: A method for providing a graphical user interface as recited in claim 9, wherein the proxy functions to enable each component which makes up the application program to be modified in a manner that is transparent to the renderer.

146. Beranek-Pleyer-Carter teaches Claims 3 and 11.

147. Beranek-Pleyer-Carter teaches that the proxy enables each component which makes up the application program to be modified in a manner transparent to the renderer. As discussed in Section VII, under an alternative construction, I have applied this limitation to mean that the proxy's component functionality is "transparent to the renderer" such that the renderer can see through the proxy to additional control options that may be available. Ex-1001, 5:12-14.

148. While Beranek explains that its renderer is unaware of how the web document is provided from the proxy, a POSITA would have been motivated for Beranek-Pleyer's proxy to function so that each component which makes up the application program is transparent to the renderer, such that the renderer can see through the proxy to additional control options that may be available, as further detailed in Carter. Carter teaches a constructor 3120 which "locates any packages (or classes in a package)." Ex-1007, 5:27-37. Carter's constructor program determines "which classes are available and then construct[s] a graphical user interface ***to reflect all of the classes***." *Id.*, 2:56-57. Figure 3 shows a command class library which stores multiple command classes 3100-3400 that are additional application functionality. *Id.*, 4:27-30.

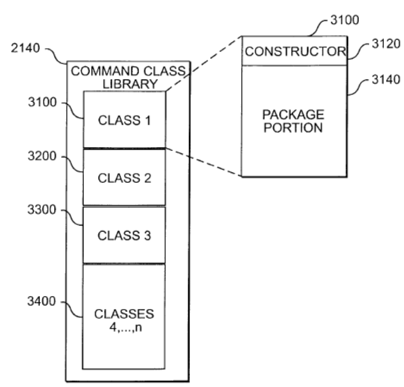


FIG. 3

149. In Figure 5B, a GUI presents four basic controls of "In Box 5210, Out Box 5220, Trash 5230, and Send Mail 5240" and additional controls for a mail

application, by instantiating one or more command classes, such as "a Spell Checker 5250 and a Thesaurus 5260." *Id.*, 5:47-56. Carter's proxy application program components are thus visible to the renderer because the proxy is transparent such that available options/classes can be seen by the renderer and can be displayed to be selected by the user. While the actual underlying details of how the software is implemented and changes made are hidden from the renderer, the control options that drive the software to make those changes are exposed to the user. As such, the proxy hides specific software details from the user but is transparent to how those details may be defined and controlled. Carter's run time activity diagrams show that successful instantiation triggers GUI reconfiguration "to add controls that make services provided by [the] command class available," and the claims recite a modification component that adds/removes visual representations for the successfully instantiated classes, precisely the exposure of additional control options to the renderer through a transparent proxy layer. In parallel, Beranek shows the same "visible options/hidden implementation" pattern: the proxy parses and rewrites incoming HTML and then passes the modified page to the browser for display; the browser merely presents the new/changed controls while remaining unaware of proxy details. Pleyer corroborates this split by implementing UI dialogs as HTML components managed by supervisory logic and DOM methods, so users see updated controls while the underlying component orchestration remains behind the proxying layer.

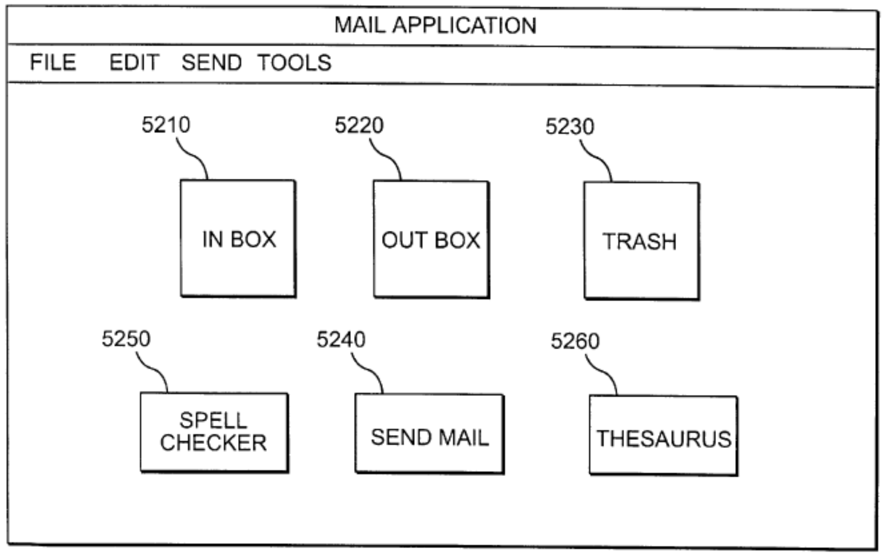


FIG. 5B

150. A POSITA would have been motivated for Beranek-Pleyer's proxy components functions to be modified in a manner that is transparent to the renderer, as taught in Carter, so that additional features can be invoked, managed, and undone through a consistent interface directly by the user which increases functionality that may be available to the user, but does not burden the renderer with the actual implementation details of those controls.

C. Ground 3: Claims 9-11 are obvious over Ditmer-Beranek

1. A POSITA would have been motivated to combine Ditmer's teachings with Beranek's teachings and would have had a reasonable expectation of success.

151. A POSITA would have been motivated to combine the teachings of Ditmer and Beranek and had a reasonable expectation of success in doing so because each relates to using a proxy to provide data in response to user requests via a browser/GUI. Ex-1008, Title, Abstract, Figure 6; Ex-1005, Title, Abstract; Ex-1006, Title, Abstract.

152. Ditmer teaches a "Web-based fault and alarm management tool that enables customers to monitor and analyze the performance of their voice and data network via a graphical user interface." Ex-1008, Abstract; *see also id.*, 3:27-36. Ditmer provides a "system and a method for communicating information relating to the dedicated voice and data networks, from an enterprise to a customer at a client workstation, the present invention includes a client browser application located at the client workstation." *Id.*, 3:52-56. As shown below in Figure 6, Ditmer teaches that a user's request is forwarded to "an integrated proxy application 640 [(purple)] for receiving and interpreting the user messages and enabling the event monitor functionality." Ex-1008, 14:31-34. Specifically, the proxy confirms incoming requests and "manages the communication of the specific customer request with the middle-tier server to actually get the request serviced." *Id.*, 23:7-14; *see also id.*, 24:14-20. The "[d]ata returned from the … server is translated back to client format, if necessary, and returned to the dispatch server as a response to the request." *Id.*, 23:14-17; *see also id.*, 9:46-50. Ditmer's "event monitor server 650 [(orange)] is

responsible for all tasks leading up to and including the management of alarms and performance reports including data collection, calculation, storage, and report generation." Ex-1008, 14:58-65.

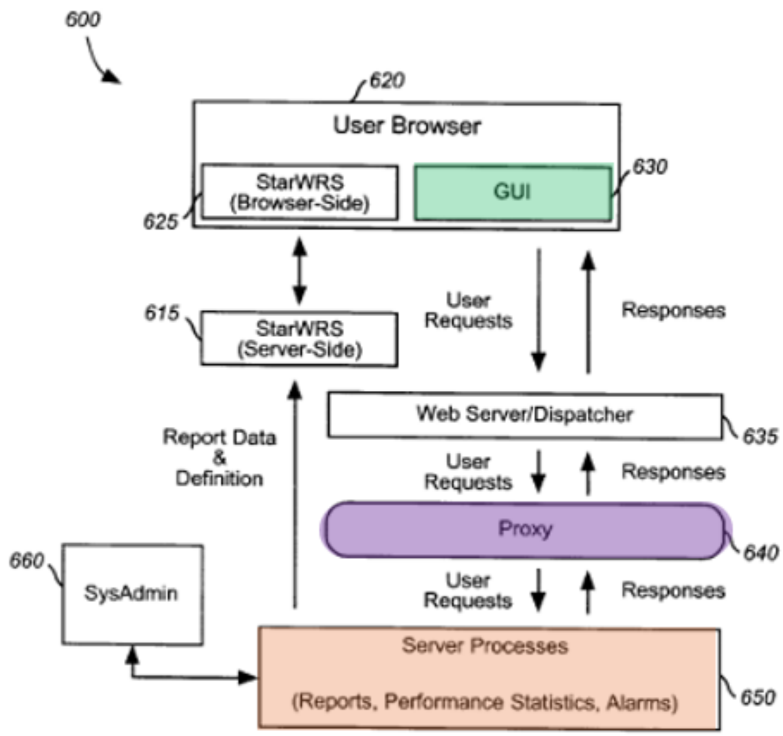


FIG. 6

153. Beranek teaches "[a] method of controlling presentation on a client of a Web document formatted according to a markup language and supported on a server, the client including a browser and connectable to the server via a computer network." Ex-1005, 15:33-36 (Cl. 1). Beranek's proxy intercepts, reformats, and controls how a Web document appears in the browser, to the user. Ex-1005, 2:54-61, 2:63-3:2. The proxy will "re-format the Web document according to some given protocol or filter property," such as "adjust[ing] or modify[ing]

foreground/background colors, overrid[ing] font types and size, control[ing] display geometry (e.g., by reserving some dedicated display area), chang[ing] URLs or display[ing] other URLs, embed files, add, remove or reconfigure frames, or control text and table formats." *Id.*, 3:5-14. The "reformatted Web document is then passed to the browser for display." Ex-1005, 3:7-8, Abstract. Beranek teaches that, as the Web document is received from the server, "the HTML is parsed to identify the format of the document and the information therein." Ex-1005, 3:2-5.

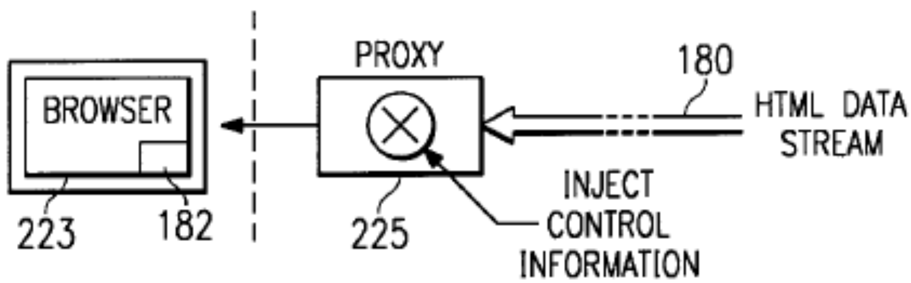


FIG. 9

154. A POSITA would have been motivated to implement Beranek's proxy technique of formatting the web document to be rendered on the user's GUI into Ditmer's proxy to ensure that the web document/data that is sent back to be rendered on the user's GUI is translated back to a format that can be displayed. While Ditmer teaches that its proxy may generate web pages based on alarm state and other data reports received from the server, Beranek explicitly teaches determining the GUI changes to be made, such as determining the GUI's new layout and updating the present layout in the GUI. Combining these teachings would ensure that the data

Ditmer returns is not only in a client format but also browser optimized (layout, frames, reserved areas, tables, fonts) in the manner Beranek describes, thereby improving clarity and usability of Ditmer's web delivered alarms and reports.

155. A POSITA would have been motivated to implement Beranek's proxy, which determines the changes to be made, into Ditmer for Beranek's added benefits of its proxy being used to "modify and/or process data before it is returned … for display on the interface," which provides a "useful tool to enhance the 'look and feel' of data displayed… and thus enables the user interface to be tailored to the Web application environment." Ex-1005, 12:16-18; *see also id.*, 14:10-13 ("optimize 'look and feel'"). Moreover, implementing Beranek's teachings provides "dynamic[] control [of] how information is presented on a… client machine" such that it enables "content modification at a client machine" while "an author of such content need only create a single version of the page." *Id.*, 2:12-14, 2:19-21. Beranek's teaching also provides "an enhanced display presentation capability" through the use of its proxy. *Id.*, 2:44-47. A POSITA would thus have been motivated to implement Ditmer with Beranek's teachings to achieve these additional benefits. Applying those benefits in Ditmer, which already streams alarms, performance data, and generated reports through a proxy to a browser GUI, would yield predictable improvements in readability, branding, and user guidance (e.g.,

reserving display areas for alert banners, reflowing tables, adjusting fonts/colors) without altering Ditmer's back-end processing.

156. A POSITA would also have been motivated to ensure that the data provided by Ditmer's process server includes one or more document viewers and associated user interface components, as taught by Beranek, so that the user may interact with the web document by providing the framework to display user interface elements, such as buttons, sliders, and sound on a web document.

157. A POSITA would have had a reasonable expectation of success in combining the teachings of Beranek and Pleyer. Moreover, both references describe methods for presenting requested data to a user in an appropriate format on a graphical user interface by the user using similar hardware and software components. Ex-1005, Title, 15:33-36 (Cl. 1); Ex-1008, Abstract. A POSITA would thus have understood that these references disclose interrelated teachings based on well-understood technologies that would have been amenable to various well-understood and predictable combinations.

2. Independent Claim 9

a. Element 9[pre]: A method for providing a graphical user interface, comprising:

158. Ditmer-Beranek teaches the preamble of Claim 9, to the extent that it is limiting.

159. Ditmer teaches a computer-implemented system for providing a graphical user interface (GUI). Ditmer teaches "an integrated Web-based graphical, user-friendly interface to receive information on the performance of their dedicated voice and data networks." Ex-1008, 3:27-36. Specifically, Ditmer provides a "system and a method for communicating information relating to the dedicated voice and data networks, from an enterprise to a customer at a client workstation, the present invention includes a client browser application located at the client workstation." *Id.*, 3:52-56.

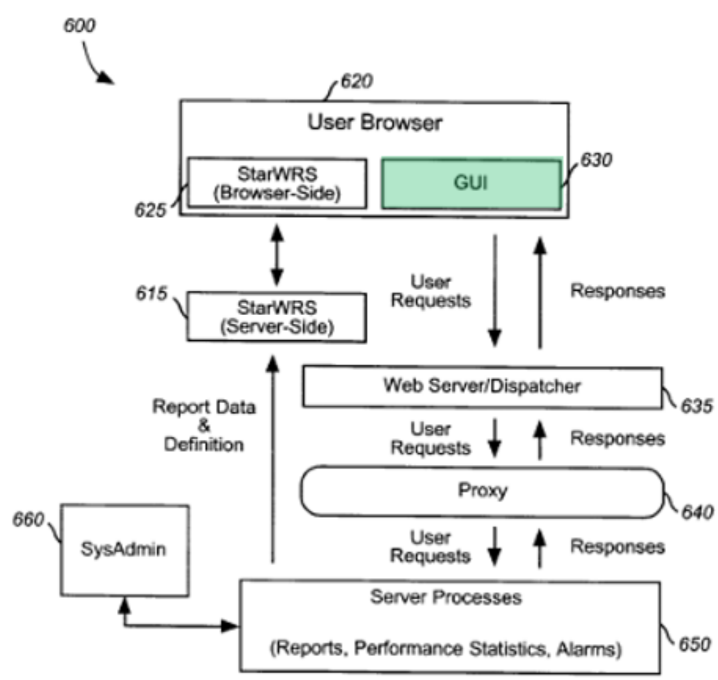


FIG. 6

b. Element 9[A]: operating a renderer defined to render a graphical user interface (GUI) for an application program;

160. Ditmer-Beranek teaches element 9[A].

161. Ditmer-Beranek teaches operating a renderer defined to render a GUI for an application program. As discussed in Section VII, I have applied this limitation as including software that renders a graphical user interface for an application program that same way as the patent, which would include software such as "a DHTML browser." Ex-1001, 4:53, 5:23-24, 7:19.

162. Ditmer-Beranek's method operates a renderer to render a GUI for the application program. Ditmer "includes a client browser application located at the client workstation," which "enables interactive Web-based communications … and also provides an integrated interface to the various enterprise application services" by invoking "a Java application/applet." Ex-1008, 3:54-62. Ditmer states that customer browser includes "***client applications responsible for presentation*** and front-end services." *Id*., 6:14-16; *see also id*., Cl. 1 ("presentation device for enabling the customer to define and display reports…"), Cl. 17 (presentation device includes a GUI and may launch directly from the client browser application). A POSITA would have understood that Ditmer's client application, responsible for presentation and front-end services on a browser, is a renderer that renders HTML pages for alarm management tools because it displays the application program on the user's workstation GUI.

163. While Ditmer teaches that its client application is a renderer, Beranek explicitly teaches a DHTML browser as a renderer, consistent with how the '428 Patent explains its renderer, as described above in Section VIII.A.2.b (element 9[A]). A POSITA would have been motivated to include Beranek's DHTML browser as Ditmer's renderer to dynamically present the user's request.

c. Element 9[B]: [(i)] operating a proxy defined to provide each component necessary for rendering the GUI to the renderer, [(ii)] wherein the proxy appears to the renderer as a single monolithic application program, and [(iii)] wherein the proxy is defined to determine changes required to be made to the GUI for the application program during execution of the application program, [(iv)] wherein a given change determined by the ox is one of rendering a new layout in the GUI and updating a present layout in the GUI; and

164. Ditmer-Beranek teaches element 9[B].

165. Ditmer-Beranek teaches operating a proxy defined to provide each component necessary for rendering the GUI to the renderer. Ditmer teaches that a user's request is forwarded to "an integrated proxy application 640 for receiving and interpreting the user messages and enabling the event monitor functionality." Ex-1008, 14:31-34. The proxy 640 "sends the generic header followed by the proxy-specific response back… to the user browser 620." *Id*., 14:42-46. Specifically, the proxy examines and confirms incoming requests to ensure that "they include validly formatted messages for the service with acceptable parameters; a translation process

for translating a message into the database query message or networking protocol; and, a management process for managing the communication of the specific customer request with the middle-tier server to actually get the request serviced." *Id.*, 23:7-14; *see also id.*, 24:14-20. The "[d]ata returned from the … server is translated back to client format, if necessary, and returned to the dispatch server as a response to the request." *Id.*, 23:14-17; *see also id.*, 9:46-50.

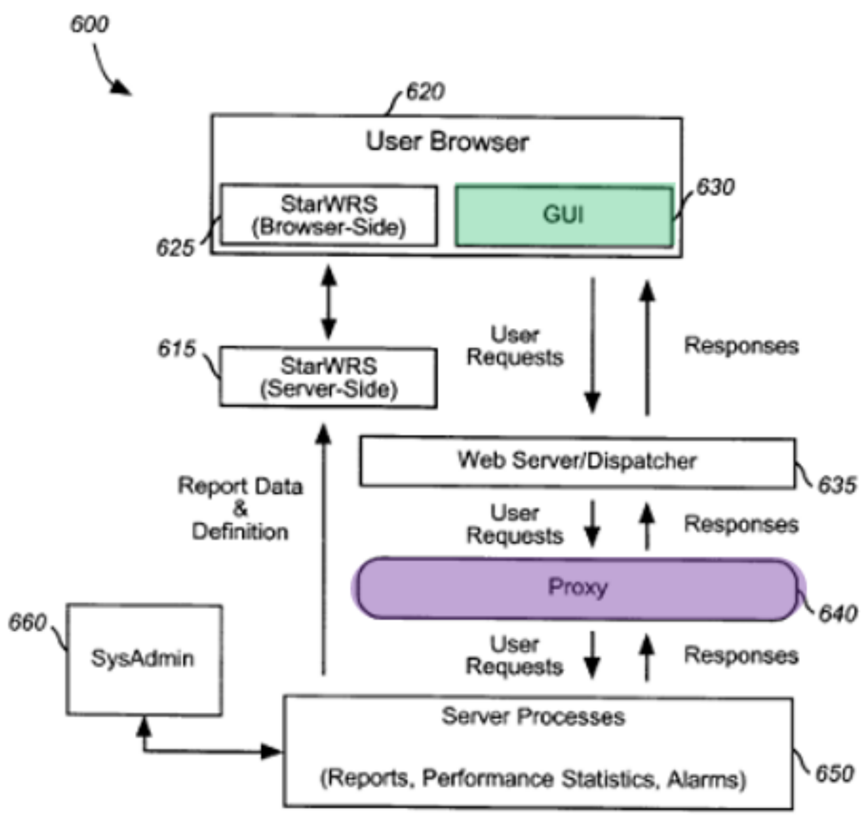


FIG. 6

166. While a POSITA would have understood that Ditmer's proxy provides each component necessary for rendering the GUI on the user's browser in response

to the user's request, Beranek provides further implementation details, as described above in Section VIII.A.2.c (element 9[B]).

167. In Ditmer-Beranek, the proxy appears to the renderer as a single monolithic application program. As discussed in Section VII, I have applied this limitation to include software that operates in the same way as "application proxy 204," which is "a single, monolithic application program and all details of the components which make up the application program are essentially hidden from renderer 200 which need only interface to application proxy 204." Ex-1001, 5:8-12. As described in Section VIII.A.2.c, Beranek's proxy 225 appears as "a single, monolithic application program and all details of the components which make up the application program are essentially hidden" from the renderer because the renderer only receives the "reformatted Web document" from proxy 225. Beranek's proxy 225 is "a piece of standalone or downloadable code that intercepts and filters the Web document retrieved over the computer network." Ex-1005, 12:14-16. Proxy 225's functions of filtering, managing/processing, and monitoring the network-retrieved data and controlling the "look and feel" of the browser interface are hidden from the renderer by being located in or behind proxy 225. Ex-1005, 8:63-67.

168. Ditmer-Beranek teaches that the proxy is defined to determine changes required to be made to the GUI for the application program during execution of the application program, where the given change is rendering a new layout in the GUI

and updating a present layout in the GUI. While Ditmer teaches that its proxy may generate web pages based on alarm state and other data, Beranek explicitly teaches determining the GUI changes to be made, such as determining the GUI's new layout and updating the present layout in the GUI, as described above in Section VIII.A.2.c (element 9[B]).

d. Element 9[C]: operating a workflow manager defined to communicate one or more document viewers and associated user interface components to the proxy for provision to the renderer as components necessary for rendering the GUI.

169. Ditmer-Beranek teaches element 9[C].

170. Ditmer-Beranek teaches operating a workflow manager defined to communicate one or more document viewers and associated user interface components to the proxy for provision to the renderer as components for rendering the GUI. As discussed in Section VII, I have applied this limitation to include software that operates in the same way as "workflow manager 216," which "manages communication between document viewers 212, user interface components 208 and, through application proxy 204, with renderer 200." Ex-1001, 5:57-59, Figure 2.

171. Ditmer-Beranek teaches operating a workflow manager. Ditmer's "event monitor server 650 is responsible for all tasks leading up to and including the management of alarms and performance reports including data collection,

calculation, storage , and report generation." Ex-1008, 14:58-65. As shown in Figure 6 below, event monitor server 650 (orange) is a "workflow manager" because it is a software module that provides report data and definition to the proxy 640 (purple) in response to the user requests for reports. *Id*., 14:54-57.

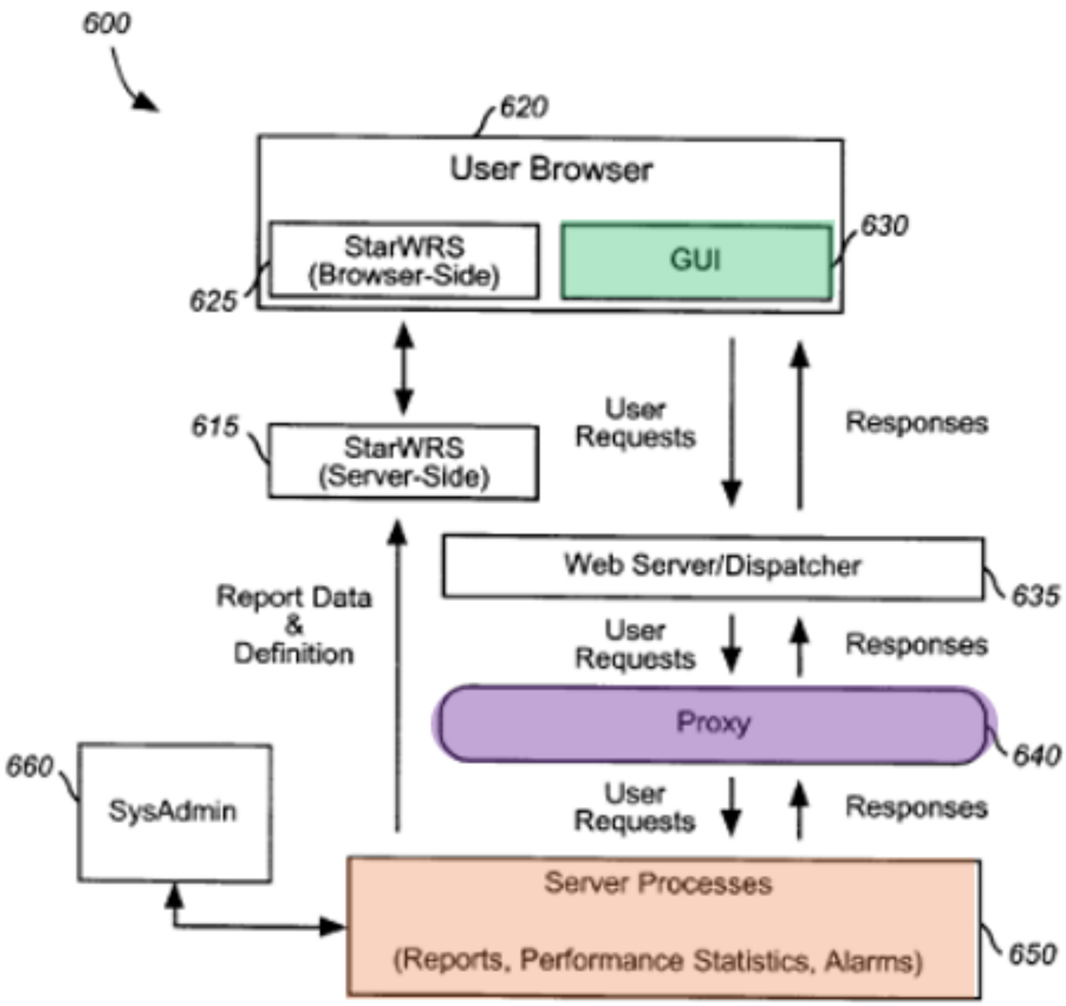


FIG. 6

172. A POSITA would have been motivated for the Ditmer's event monitor server 650 to communicate one or more document viewers and associated user

interface components to the proxy, as taught in Beranek. Beranek's Web document from the server includes one or more document viewers. The '428 specification explains that a document viewer includes "viewers appropriate to observe and interact with data appropriate to the application program, in all of its possible contexts," such as including an "'Edit Picture' viewer, a 'Special Effects' viewer, an 'Object Manager' viewer, etc." Ex-1001, 5:34-41. Beranek teaches that, as the Web document is received from the server, "***the HTML*** is parsed to ***identify the format of the document and the information*** therein." Ex-1005, 3:2-5. The proxy will "***re-format*** the Web document according to some given protocol or filter property," such as "adjust[ing] or modify[ing] foreground/background colors, overrid[ing] font types and size, control[ing] display geometry (e.g., by reserving some dedicated display area), chang[ing] URLs or display[ing] other URLs, embed files, add, remove or reconfigure frames, or control text and table formats." *Id.*, 3:5-14. A POSITA would have understood that Beranek teaches a document viewer that communicates the Web document to be rendered via the proxy, which adjusts and modifies the display geometry and adds/removes/reconfigures frames so that the data to be observed and interacted with in all its possible contexts. Beranek teaches that the viewer content delivered via the proxy is the Web document itself, HTML that the proxy tailors and supplies to the browser, so the browser renders the appropriate "viewer" for the data (e.g., reserved display areas, frames, tables, embedded items). Beranek further identifies UI components directly: the proxy can reformat or inject controls such as "checkbox, button, [and] pulldown menu" and arrange content into tables/frames, thereby supplying the renderer with the associated user interface components alongside the document viewer. Ex-1005, 9:8-51. In combination, Ditmer's server-side generation/selection of report pages and alarm views communicated via proxy, and Beranek's proxy-mediated delivery of the concrete HTML viewer and controls to the browser, teach "communicat[ing] one or more document viewers and associated user interface components to the proxy for provision to the renderer."

173. Beranek's Web document from the server includes associated user interface components. The '428 specification explains that user interface components "include controls such as button, sliders, radio buttons, edit boxes, etc., can be implemented in a variety of manners, including as ActiveX controls, Java applets, etc." Ex-1001, 4:11-14. As described above, Beranek's proxy will adjust the "look and feel" of a Web document, which is formatted according to a markup language, such as HTML. Specifically, Beranek's proxy will "re-format the Web document according to some given protocol or filter property," such as "adjust[ing] or modify[ing] foreground/background colors, overrid[ing] font types and size, control[ing] display geometry (e.g., by reserving some dedicated display area), chang[ing] URLs or display[ing] other URLs, embed files, add, remove or

reconfigure frames, or control text and table formats." Ex-1005, 3:5-14. Beranek's proxy may also "change the font of the data, or to ***add sounds, or to arrange data in a certain tabular format***, or the like." *Id.*, 11:42-45. Additionally, Beranek teaches that its proxy's filter mechanism is able to reformat a "***checkbox, button, pulldown menu or the like***," and ***images*** on the web document. *Id.*, 9:8-51. A POSITA would have understood that Beranek's Web document, formatted according to HTML, includes associated user interface components within the document viewer so that the user may interact with the Web document because HTML is a common language used to structure content and provides the framework to display user interface elements, such as buttons, sliders, and sound. That is a POSITA would have understood that Beranek explains that the proxy can adjust "look and feel," control display geometry, add/remove/reconfigure frames, embed files/sounds/images, and modify text/table formats, and even rewrite FORM/INPUT elements (e.g., checkboxes, buttons, pull-downs), so the browser receives a page that already includes the interactive UI components to operate the viewer. And then, when paired with Ditmer's proxy pathway and report/alarms workflow server, the resulting system operates a workflow manager that communicates the relevant viewer pages plus their UI controls to the proxy, and the proxy in turn provides them to the browser renderer as the components necessary to render the GUI.

3. Dependent Claims 10 and 11

a. Claim 10: A method for providing a graphical user interface as recited in claim 9, wherein the proxy functions to hide details of each component which makes up the application program from the renderer.

174. Ditmer-Beranek teaches Claim 10.

175. Ditmer-Beranek teaches that the proxy hides details of each component which makes up the application program from the renderer. As discussed in Section VII, I have applied this limitation to include that the proxy's component functionality "is not visible to the user." Ex-1001, 5:23-25.

176. Ditmer-Beranek teaches that its proxy hides details of its components from the renderer as explained in Section VIII.B.2.c (element 9[b]). Moreover, for the same reasons described in Section VIII.A.3.a, Beranek's proxy 225 appears just as "application proxy 204," which is "a single, monolithic application program and all details of the components which make up the application program are essentially hidden from renderer 200 which need only interface to application proxy 204." Ex-1001, 5:8-12.

b. Claim 11: A method for providing a graphical user interface as recited in claim 9, wherein the proxy functions to enable each component which makes up the application program to be modified in a manner that is transparent to the renderer.

177. Ditmer-Beranek teaches Claim 11.

178. Ditmer-Beranek teaches that the proxy enables each component which makes up the application program to be modified in a manner transparent to the renderer. As discussed in Section VII, I understand that PO stated during a meet-and-confer that it interpreted "transparent to the renderer" as requiring that the renderer is not aware of any modification of "each component which makes up the application program." For the reasons I explained in Section VIII.A.3.b (claims 3 and 11), Beranek teaches that its proxy enables each component which makes up the application program to be modified in a manner that is transparent to the renderer in PO's alternative construction.

IX. CONCLUSION

179. For the reasons set forth above, I believe that Claims 1-5 and 9-13 of the '428 Patent are unpatentable in view of the prior art.

180. In signing this declaration, I recognize that it will be filed as evidence in a contested case before the Patent Trial and Appeal Board of the United States Patent and Trademark Office. I also recognize that I may be subject to cross-examination in the case and that cross-examination will take place in the United States. If cross-examination is required of me, I will appear for cross-examination within the United States during the time allotted for cross-examination.

181. I reserve the right to supplement my opinions in the future to respond to any arguments that the Patent Owner raises and to take into account new information as it becomes available to me.

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

Dated: November 7 2025

Respectfully submitted,

Brian W. Anthony, Ph.D.