

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

CISCO SYSTEMS, INC.,
Petitioner,

IPR2026-00209
Patent No. 11,576,046

**PETITION FOR *INTER PARTES* REVIEW
UNDER 35 U.S.C. § 312 AND 37 C.F.R. § 42.104**

TABLE OF CONTENTS

Table of Contents	2
Petitioner’s Exhibit List	4
I. Introduction.....	7
II. Grounds for Standing.....	9
III. Note.....	10
IV. Summary of the ’046 patent	10
V. Effective priority date	10
VI. Prosecution history	11
VII. Level of ordinary skill in the art	12
VIII. Background.....	13
IX. Claim Construction.....	15
X. Relief requested and reasons therefore.....	15
XI. Identification of how the claims are unpatentable.....	16
A. Challenged Claims	16
XII. Statutory grounds.....	16
A. Statement of Material Facts.....	17
B. Ground 1	19
1. Abuan	19
2. Independent Claim 1	21
3. Independent Claim 11	48
C. Ground 2.....	53
1. Ludwig	53
2. Reasons to combine Abuan and Ludwig.....	57

3.	Dependent claims 5 and 18	58
D.	Ground 3	61
1.	Lawson	61
2.	Reasons to combine Abuan and Lawson	63
3.	Dependent claims 7 & 12	64
E.	Ground 4	67
1.	Guzman	67
1.	Reasons to combine Abuan and Guzman	69
2.	Dependent claims 8 and 13	71
XIII.	Conclusion	78
XIV.	Mandatory notices	79
A.	Real party-in-interest	79
B.	Related matters	79
C.	Lead and back-up counsel and service information	79
	Claims Appendix	81
	Certificate of Word Count	86
	Certificate of Service	87

PETITIONER’S EXHIBIT LIST

Ex.1001	U.S. 11,576,046
Ex.1002	Prosecution History of U.S. 11,576,046
Ex.1003	Declaration of Bruce McNair under 37 C.F.R. § 1.68
Ex.1004	<i>Curriculum Vitae</i> of Bruce McNair
Ex.1005	U.S. 2011/0249077 to Abuan et al.
Ex.1006	U.S. 2012/0092438 to Guzman Suarez et al.
Ex.1007	U.S. 2013/0019297 to Lawson et al.
Ex.1008	U.S. 5,689,641 to Ludwig et al.
Ex.1009	D. E. Knuth, Selected Papers on Computer Science (Cambridge Univ. Press), 1996
Ex.1010	B. W. Boehm, Software Engineering Economics (Prentice-Hall, Inc.), 1981
Ex.1011	U.S. 5,339,431 to Rupp et al.
Ex.1012	U.S. 5,615,400 to Cowsar et al.
Ex.1013	U.S. 5,946,486 to Pekowski
Ex.1014	U.S. 6,243,764 to Leach et al.
Ex.1015	U.S. 5,802,367 to Held et al.
Ex.1016	U.S. 5,218,697 to Chung et al.
Ex.1017	U.S. 5,498,003 to Gechter
Ex.1018	U.S. 6,163,858 to Bodamer
Ex.1019	U.S. 2012/0019610 to Hornyak et al.
Ex.1020	Complaint, <i>Damaka, Inc. v. Cisco Systems, Inc.</i> , No. 2:25-cv-00593 (E.D. Tex. Sept. 8, 2025).
Ex.1021	U.S. 2008/0178264 to Keohane et al.
Ex.1022	U.S. 2005/0235044 to Tazuma
Ex.1023	U.S. 2006/0037064 to Jeffries et al.

Ex.1024	<i>Reserved</i>
Ex.1025	<i>Reserved</i>
Ex.1026	<i>Reserved</i>
Ex.1027	<i>Reserved</i>
Ex.1028	<i>Reserved</i>
Ex.1029	U.S. 2009/0193394 to Dasch et al.
Ex.1030	U.S. 2008/0022327 to Murray et al.
Ex.1031	U.S. 2002/0199019 to Battin et al.
Ex.1032	U.S. 6,101,607 to Bachand et al.
Ex.1033	U.S. 6,311,283 to Gonzalez
Ex.1034	U.S. 6,295,048 to Ward et al.
Ex.1035	U.S. 2009/0310023 to Nakayama
Ex.1036	U.S. 2008/0127183
Ex.1037	U.S. 7,574,202
Ex.1038	U.S. 7,461,144
Ex.1039	U.S. 2007/0185998
Ex.1040	Prosecution History of U.S. 9,270,744
Ex.1041	IEEE 100, The Authoritative Dictionary of IEEE Standard Terms, 7 th Ed. (IEEE Press), 2000
Ex.1042	A. Silberschatz et al., Operating System Concepts, 8 th Ed. (John Wiley & Sons), 2009
Ex.1043	U.S. 6,820,235
Ex.1044	U.S. 2009/0106830
Ex.1045	U.S. 5,835,435
Ex.1046	U.S. 5,742,641
Ex.1047	Plaintiff's Disclosure of Asserted Claims and Infringement Contentions, <i>Damaka, Inc. v. Cisco Systems, Inc.</i> , No. 2:25-cv-00593 (E.D. Tex. Nov. 13, 2025).

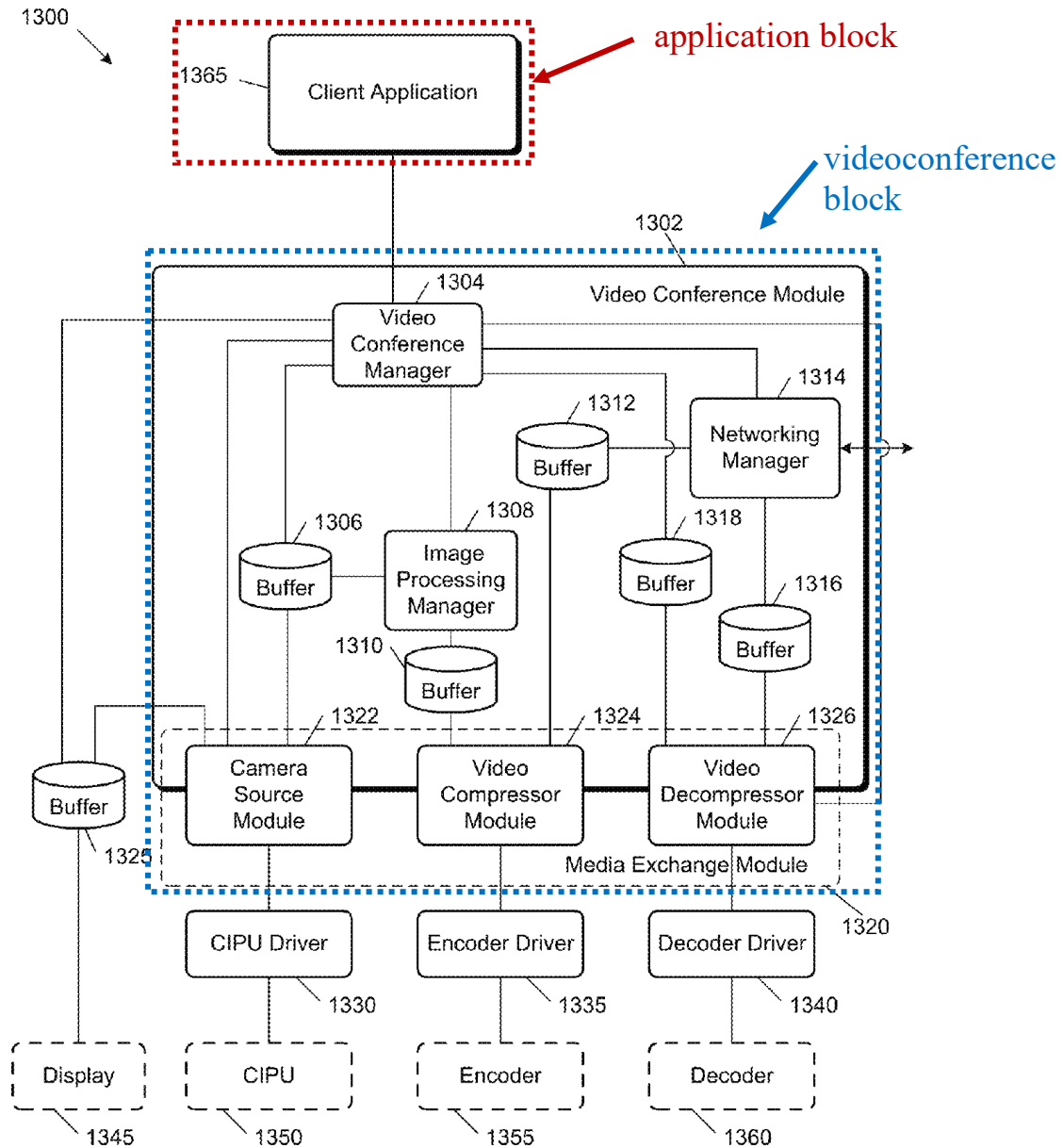
Ex.1048	IBM 1130 materials (Scientific Subroutine Package manual noted) (“IBM 1130”)
Ex.1049	Alfred Aho and Jeffrey Ulman, “Principles of Compiler Design” (1977)
Ex.1050	Request for Comments (RFC) 793 (Sept. 1981)
Ex.1051	Request for Comments (RFC) 2616 (June 1999)
Ex.1052	Request for Comments (RFC) 3261(June 2002)

I. Introduction

Pursuant to 35 U.S.C. §§ 311, 314(a), and 37 C.F.R. § 42.100, Cisco Systems, Inc. (“Petitioner”) respectfully requests that the Board review and cancel as unpatentable under 35 U.S.C. §103 claims 1-20 (hereinafter, the “Challenged Claims”) of U.S. 11,576,046 (the “’046 patent,” Ex.1001).

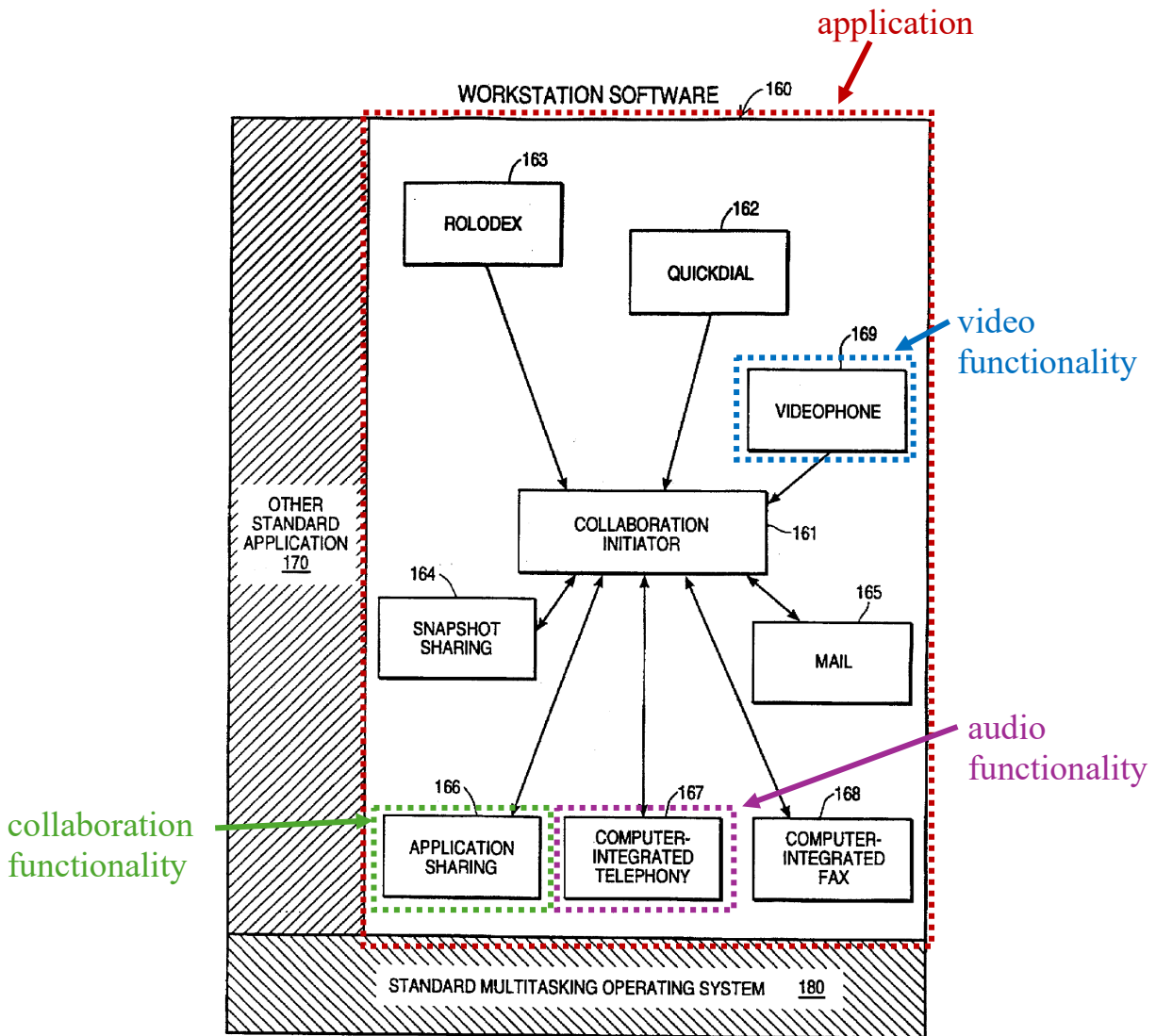
The ’046 patent is directed to “improving functionality in software applications.” Ex.1001, Abstract. The patent acknowledges that one known way to improve software was to rework an existing application to add “desired functionality” by incorporating additional executable instructions that “provide such additional functionality.” Ex.1001, 3:49-55. The claimed innovation does the same thing, except that the added instructions are characterized as being a “function block” that is added to a “superblock.” Ex.1001, 4:37-46. The superblock “may be an application,” while the function block provides capabilities for “audio, video, collaboration..., and/or other functions.” Ex.1001, 3:3-5, 4:46-55.

Organizing computer software into blocks of code was not new. The prior art figure below (dating from 2010) shows an application block designed to use video functions—specifically, videoconference functionality—provided by a separate block. Ex.1005, FIG. 13.



Ex.1005, Fig. 13 (annotated).

Nor was the modular design of software applications in any way “new” even in 2010. The figure below, dating from 1993, shows the modular design of an application with discrete blocks providing audio, video, and collaboration functionality. Ex.1008, FIG. 20.



Ex.1008, Fig. 20 (annotated).

Because the Challenged Claims merely describe concepts and techniques that were already known, Petitioner asks the Board to institute trial and find the claims unpatentable.

II. Grounds for Standing

Petitioner certifies that the '046 patent is eligible for IPR and that Petitioner is not barred or estopped from requesting IPR of the Challenged Claims.

III. Note

Petitioner cites to exhibits' original page numbers. **Emphasis** in quoted material has been added. Claim terms are presented in *italics*.

IV. Summary of the '046 patent

The '046 patent purports to describe an improved “manner in which functionality is accessed in certain environments, such as mobile device environments.” Ex.1001, 1:66-67. The improvement consists of a modular approach to software using a “superblock application” and a “function block” that provides “one or more functions (e.g., capabilities)” to the superblock. Ex.1001, 4:43-45. The function block comprises one or more blocks of executable instructions. Ex.1001, 5:18-20. The superblock may access the function block instructions by calling an application programming interface (API). Ex.1001, 6:30-36. Ex.1003, ¶¶33-37.

V. Effective priority date

The '046 patent is based on an application filed December 2, 2020 and claims priority to an earlier application filed on September 11, 2013. It also claims the benefit of a provisional application filed July 16, 2013. Determining whether the '046 patent is entitled to an earlier priority date is unnecessary in this proceeding, however, as all the cited and relied-upon prior art predates July 2013. Accordingly, Petitioner has not undertaken a priority date analysis. Petitioner does

not waive any right or opportunity it may have to dispute the priority date of the '046 patent in this or another forum where the issue is relevant.

VI. Prosecution history

The '046 patent is the seventh patent issued in a priority chain dating to July 2013. Ex.1002, 79. The same examiner who was assigned to the other six patents in the priority chain allowed the '046 patent without any prior art rejections, after a few examiner-initiated interviews and proposed amendments. Ex.1002, 84.

The Application from which the '046 patent issued was filed with a single claim, which was identical to claim 1 of U.S. Patent No. 9,578,092 (the '092 patent). Ex.1002, 11, 87. Before issuing an Office action, the examiner contacted the applicants to “propose[] some amendment to distinguish the claim to address the statutory double patenting [sic].” Ex.1002, 89. The Examiner indicated that a terminal disclaimer “would likely be needed thereafter.” Ex.1002, 89. The applicants declined the examiner’s invitation, asking for a formal Office action. The examiner sent the first and only Office action a week later, rejecting the claim for double patenting. Ex.1002, 86-87.

In response to the Office action’s double patenting rejection, the applicants canceled claim 1 and presented a fresh set of 20 claims. Ex.1002, 595-599. The new independent claims largely removed functional limitations relative to the originally filed claim, but they included a new limitation reciting that a function is

performed “without switching context from the superblock application to another application on the electronic device.” Ex.1002, 596. The applicants had presented claims with similar features in a related case (U.S. Patent No. 9,270,744), which issued without any prior art rejections. Ex.1040, 37.

Two months after receiving the applicants’ response, the examiner contacted the applicants requesting a terminal disclaimer, and proposing a minor amendments to address potential issues under 35 U.S.C. § 112. Ex.1002, 620. After receiving the terminal disclaimer the examiner sent a Notice of Allowability for all claims. Ex.1002, 612-619. Similar to the ’744 patent, the Examiner’s statement of reasons for allowance emphasized the independent claim limitations relating to certain steps being performed “without switching context from the superblock application to another application.” Ex.1002, 618.

VII. Level of ordinary skill in the art

A Person of Ordinary Skill in The Art (“POSITA”) in July 2013 would have had a working knowledge of the software programming art that is pertinent to the ’046 Patent, including the use of software modules and application programming interfaces (APIs). A POSITA would have had a bachelor’s degree in computer science, computer engineering, or an equivalent, and three years of professional experience relating to computer programming. Lack of professional experience can be remedied by additional education, and vice versa. Ex.1003, ¶¶18-20.

VIII. Background

As of the July 2013 priority date of the '046 patent, computer science and software programming had been developing as technical fields of study for at least forty years—and, same say, for several millennia. Ex.1009, 5, 185. Early computers were programmed using punch cards onto which numbers were encoded. The computer interpreted the numbers as instructions. Ex.1009, 227-229. Soon afterwards, the programmers began creating programs that would transform more human-friendly instructions into computer-executable punch cards—a process that came to be known as “compiling.” Ex.1009, 231 (describing the “famous IT compiler”); Ex.1049, 518-519.

Computers rapidly grew in their complexity, with new instructions and larger memories making it possible to tackle more sophisticated problems. Punch cards gave way to perforated tapes and then “teletype” terminals, which combined a keyboard for providing input to a computer with a display screen for displaying the computer’s output.

As computers grew more sophisticated, so too did programs expand in their complexity. By at least the 1980s, programmers could readily access “standard functions” provided as part of programming languages such as FORTRAN, COBOL, and Pascal. Ex.1010, 654; Ex.1048. Even more extensive functions were available as “Program Libraries,” which would “typically [be] called as

subroutines” from a program. Ex.1010, 654.

To facilitate the use of these additional “libraries,” the technology for “compiling” a program was split into a two-step process. In a first step, human-readable instructions in a single file were converted into computer-executable “object code.” Ex.1011, Abstract. In a second step, the various sources of instructions (both application source files and program libraries) were integrated into a complete stand-alone application. Ex.1011, 1:24-28; Ex.1012, 1:25-29. This second step was known as “static linking.” Ex.1012, 1:29; Ex.1013, 1:52-54, 1:66-2:2; Ex.1014, 2:5-17. With static linking, the “static-link library has access to the client program code and data,” that is, “static-link libraries have access to the client program execution context.” Ex.1015, 1:18-27, 1:39-40.

As software became even more complex, programmers recognized the value of allowing commonly used functions to be shared among multiple programs. Sharing allows multiple programs to call the same functions but avoids having each program incorporate its own redundant copy of the call functions. Technology for this came to be known as a “dynamic-link library. “In contrast to static-link libraries, several client programs can share a single copy of the code contained in a DLL, which saves memory space.” Ex.1015, 1:27-29. Just like static-link libraries, “DLLs are executed in the context of the calling code; that is, the DLL can access the code, data space, and the stack space of the client program that invoked a

function contained in the DLL.” Ex.1015, 1:35-38. As a result, both static- and dynamic-linked libraries provide functions that can be executed without switching from the context of the calling program. These concepts were well-known in the computer programming art long before the ’046 patent’s earliest priority date. Ex.1016, 6:26 (“DLLs are a well-known feature...and are specific instances of APIs which are available as common resources to any application or other type of executable code”); Ex.1017, 16:33-34 (“Procedures to allow an application to incorporate new DLL's are well known in the art.”); Ex.1018, 6:65-66 (“Two forms of linking are common: static linking and dynamic linking.”).

IX. Claim Construction

Claim terms in IPR are construed according to their “ordinary and customary meaning” to those of skill in the art. 37 C.F.R. § 42.100(b). For the purposes of this proceeding and the grounds presented herein, Petitioner proposes that no terms require express construction. *Nidec Motor Corp. v. Zhongshan Broad Ocean Motor Co.*, 868 F.3d 1013, 1017 (Fed. Cir. 2017).

X. Relief requested and reasons therefore

Petitioner asks that the Board institute a trial for *inter partes* review and cancel the Challenged Claims in view of the analysis below.

XI. Identification of how the claims are unpatentable

A. Challenged Claims

Petitioner challenges claims 1-20.

Patent Owner asserts claims 1-4, 7-15, 17, and 19-20 in the co-pending litigation. *See* Ex.1020, ¶49 (asserting “at least claim 20”); Ex.1047, 2. A finding that the Challenged Claims (1-20) are unpatentable in this proceeding will resolve the parties’ dispute in the co-pending litigation and obviate any need for a trial regarding the ’046 patent, substantially reducing the time and expense of that litigation for all parties.

XII. Statutory grounds

Grounds	Claims	Basis
#1	1-4, 6, 9-11, 14-17, 19, and 20	35 U.S.C. § 103 over Abuan
#2	5, 18	35 U.S.C. § 103 over Abuan and Ludwig
#3	7, 12	35 U.S.C. § 103 over Abuan and Lawson
#4	8, 13	35 U.S.C. § 103 over Abuan and Guzman

U.S. 2011/0249077 to Abuan et al. (Ex.1005, “**Abuan**”) published on October 13, 2011.

U.S. 2012/0092438 to Guzman Suarez et al. (Ex.1006, “**Guzman**”) published on April 19, 2012.

U.S. 2013/0019297 to Lawson et al. (Ex.1007, “**Lawson**”) published on

January 17, 2013.

U.S. 5,689,641 to Ludwig et al. (Ex.1008, “**Ludwig**”) published on November 18, 1997.

Abuan, Guzman, Lawson, and Ludwig are prior art under 35 U.S.C. § 102(a)(2).

Petitioner’s § 103 obviousness grounds rely on the combined teachings of the references and not on a physical incorporation of elements. *See In re Mouttet*, 686 F.3d 1322, 1332 (Fed. Cir. 2012).

Petitioner also cites below to additional prior art as evidence of the background knowledge of a POSITA and to provide contemporaneous context to support Petitioner’s assertions regarding what a POSITA would have understood from the prior art in the grounds. *See Yeda Research v. Mylan Pharm. Inc.*, 906 F.3d 1031, 1041-1042 (Fed. Cir. 2018) (affirming the use of “supporting evidence relied upon to support the challenge”); 37 C.F.R. § 42.104(b); *see also K/S HIMPP v. Hear-Wear Techs., LLC*, 751 F.3d 1362, 1365-66 (Fed. Cir. 2014); *Arendi S.A.R.L. v. Apple Inc.*, 832 F.3d 1355, 1363 (Fed. Cir. 2016).

A. Statement of Material Facts

1. As of the priority date, it was common for a server to require a user login to access the server’s services. Ex.1043, Fig. 6 (showing a “typical login screen” asking for “an account name 293 and a password 295”); Ex.1021, [0024]

(“A user login typically includes a user name and password.”); Ex.1022, [0028]

(describing “operations typically performed” during login for “authentication and authorization”); Ex.1023, [0002] (describing the “well known” login process in which “a server computer is responsible for granting access to a remote user based on a valid combination of userID and password”).

2. The use of APIs to define how an application accesses functionality of a software module was well-known and commonly used. Ex.1029, [0087] (“APIs are well-known in the art”); Ex.1030, [0038] (“One of the primary purposes of an API is to provide a set of commonly-used functions”); Ex.1031, [0040] (“A well-known API”); Ex.1032, 5:32-48 (“An API is a well-known programming device.... [C]alls to API functions 60 are included in application program 56; the programmer is relieved of any task of having to incorporate or integrate the code itself of functions 60 in application program 56”).

3. A POSITA would have been familiar with device drivers for output devices and would have recognized that they commonly included APIs for reporting on an output device’s capabilities. Ex.1033, 6:30-32 (using “software (e.g., the device drivers) to determine the capabilities” of hardware resources); Ex.1034, 11:8-26 (describing device drivers determining a “display device’s maximum resolution”); Ex.1035, Abstract (“determine capabilities of graphics hardware from a device driver”), [0004] (“provid[ing] a set of application

programming interfaces (APIs) to ascertain device contexts and determine capabilities of graphics hardware from a device driver”); Ex.1036, [0033] (“device capabilities ... may be exposed through one or more APIs”).

4. Allowing unrestricted and unlimited use of a server-provided feature would create the undesirable risk of the server becoming overwhelmed and failing, either intentionally (in a denial-of-service attack) or unintentionally. Ex.1037, 19:5-10 (“denial of service attack (also, DoS attack) is an attack... that causes a loss of service to users, typically... by consuming the bandwidth of the victim network or overloading the computational resources of the victim system”); Ex.1038, 2:30-39 (“denial of service (DOS) attacks, where... the server at the target IP address is overwhelmed by fake requests for service, and is either unable to service ‘real’ requests for service, or crashes due to overload”); Ex.1039, [0002] (“a denial of service attack, one or more attackers may send packets towards one or more servers, overwhelming the servers so that they are no longer able to serve their legitimate clients.”).

B. Ground 1

1. Abuan

Abuan generally relates to the field of communications software for mobile devices, and more particularly, to real-time video conferencing from mobile devices. Abuan describes a modular approach to facilitating such conferencing

features, with functionality organized into components such as a video processing and encoding module, media exchange module, video processing module, video conferencing and processing module, and many others. Ex.1005, [0043], [0099], [0056]. The modules' functionalities are accessed via "a set of application program interfaces (APIs)." Ex.1005, [0044]; Ex.1003, ¶¶43-48.

Abuan is in the same field of endeavor as the '046 patent, namely, the "manner in which functionality is accessed in certain environments, such as mobile device environments." Ex.1001, 1:66-67; *see* Ex.1005, [0001]-[0002] (describing ways that "functionality" is accessed on "portable devices, such as smartphones"). Abuan and the '046 patent also both generally relate to implementing functionality via software. *See* Ex.1001, Abstract ("providing additional functionality to existing software"); Ex.1005, [0006]-[0007], [0105] (describing how instant messaging or voice-over-IP applications "use the video conferencing functions of the video conference module"). Abuan and the '046 patent both describe at length features for facilitating videoconferencing. Ex.1001, Abstract, 4:43-51 (purpose of a function block is to provide functions including "audio, video,... conferencing, meetings, and/or other functions"); Ex.1005, [0099] (describing a "video conference module... for performing a variety of video conferencing functions"). Ex.1003, ¶49.

Abuan is also reasonably pertinent to a problem allegedly addressed by the

'046 patent. The '046 patent describes purported problems associated with using multiple applications on a mobile device, potentially requiring a user to “switch[] back and forth between the video window of [a] call and [a] superblock application.” Ex.1001, 4:22-26. The '046 patent’s proposed solution is to provide functionality—such as communication functions—to a superblock application. Ex.1001, 4:43-61. Abuan similarly describes how communication functions are provided to a client application, which may be a “stand-alone application” or may itself be further “integrated into another application.” Ex.1005, [0105]. Ex.1003, ¶50.

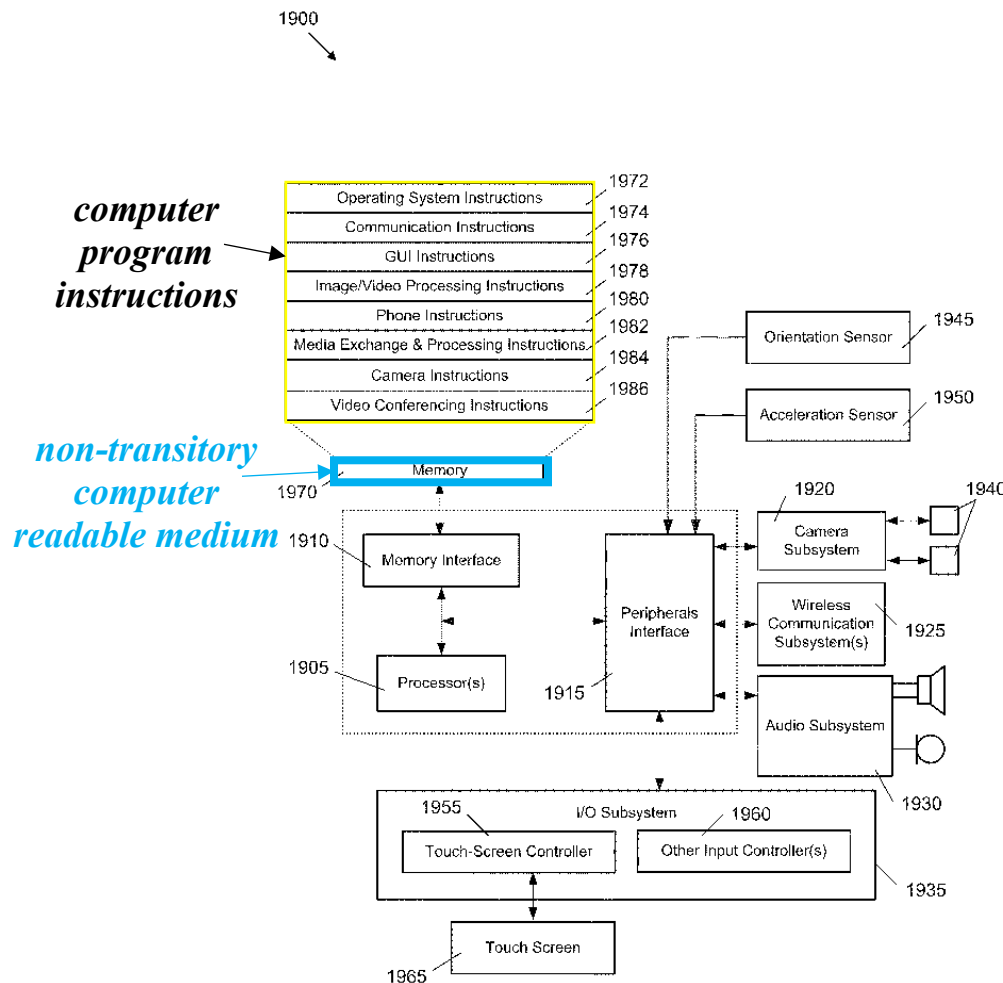
2. Independent Claim 1 and dependent claims

[1.0] A non-transitory computer readable medium embodying a computer program comprising instructions that include a function block and a superblock application,

Abuan teaches a “method for managing a video conference between a first device and a second device,” implemented as software processes. Abuan, Abstract; [0235]. The disclosed program code is embodied as computer instructions residing in read-only memory or stored on magnetic media and loaded into memory for execution, and may be organized as sub-parts of a larger program, separate programs, or combinations thereof. Ex.1005, [0234]. The modules are comprised of software instructions, such as video conferencing instructions 1986, that are stored in memory. Ex.1005, [0233], [0254], Fig. 19; *see also* [0254] (explaining

how an “API-implementing component” may be a “module”); Ex.1019, [0032]

(“programmable elements, often referred to as modules” “may include a program, a subroutine, a portion of a program, or a software component”). Abuan further confirms the software is stored on a machine-readable medium in a computer-readable form. Ex.1005, [0248].

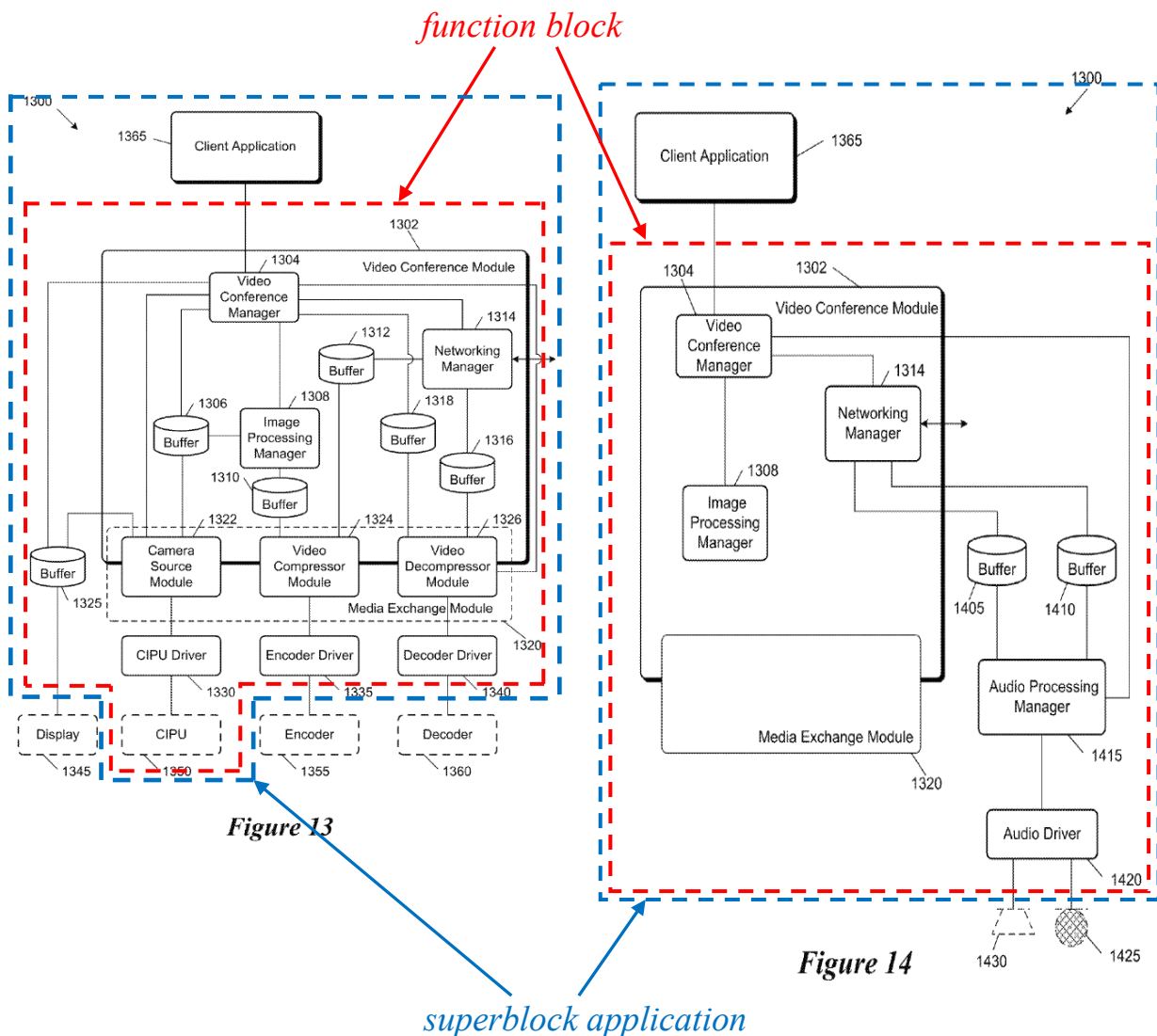


Ex.1005, FIG. 19 (annotated).

Abuan’s “memory” storing instructions for the video conference client application is *a non-transitory computer-readable medium embodying a computer*

program comprising instructions. Ex.1003, ¶¶51-53.

Abuan's video conference software includes a "client application 1365," a "video conference module 1302," and other software modules (e.g., video compressor and decompressor modules) that are used by the mobile device for video conferencing. Ex.1005, [0099], [0101].



Ex.1005, Figs. 13 and 14 (annotated).

The various sub-modules of the video conference module 1302 perform

various video conference functions “such as image processing, video conference management, and networking.” Ex.1005, [0101]. The client application 1365 may be a “stand-alone” application like “Skype,” which receives commands or selections from the user of the mobile device, and “sends instructions to the video conference module” to conduct a video conference. Ex.1005, [0172], [0099], [0101], [0105]. Ex.1003, ¶¶54-55.

As illustrated in annotated Figs. 13 and 14 above, Abuan’s video conference module 1302 and other modules (media exchange module 1320, CIPU driver 1330, CIPU 1350, Audio Driver) outlined in red correspond to the claimed *function block*. The combination of the *function block*¹ modules with the client application 1365 corresponds to the claimed *superblock application*, as outlined in blue.

¹ For simplicity, this Petition occasionally refers to “video conference module 1302” and the “video conference modules” to collectively describe the software components of the “video conferencing processing module 1300” that are outlined in dashed red lines above, including the media exchange module 1320 and various drivers (e.g., 1330, 1335, 1340, 1420). Thus, any analysis in this Petition linking Abuan’s “video conference module 1302” to the claimed “*function block*” is not intended to limit or modify the mapping provided with respect to Figs. 13 and 14 above.

Ex.1003, ¶¶56-57.

A POSITA would have understood Abuan’s “stand-alone” application to mean a single, integrated application that includes the client application functionality (e.g., user interface and video display windows), and the functional modules on which the client application relies to conduct the video conference. Ex.1003, ¶58; *see* Ex.1005, [0262], [0239]. Moreover, it would have been obvious to compile all software modules of the “video conferencing and processing module 1300”—including the client application 1365 and the video conference module 1302—together as one application, because this was a commonly-used and straightforward way to create a stand-alone application. Ex.1003, ¶¶58-60; *see* Ex.1012, 1:26-29.

[1.1] wherein the function block is accessible only to the superblock application on an electronic device, and

As explained above for limitation [1.0], Abuan’s video conferencing functional modules (*function block*) are “*accessible [] to the superblock application,*” because the client application 1365 *access*[es] the video conference module to use the video conference functionality. Ex.1005, [0171].

As also explained above for limitation [1.0], Abuan discloses or renders obvious, the client application 1365 being compiled together with the video conference module 1302 and other related functional modules into a unitary “stand-alone” software application (*the superblock application*). In the compiled

“stand-alone” application, the video conference module 1302 functionality would be “*accessible only*” to the compiled application, and not to other applications installed on the device. Ex.1003, ¶¶58-59, 62-66, 88; *see* Ex.1042, 317 (a program’s attempt to access memory outside its separate memory space is treated as a fatal error), 325.

Additionally, Abuan discloses using APIs for instructions between the client application and the video conference module 1302 to keep the underlying software code hidden, and restricting use of functional video conference modules (e.g., API-implementing components) to code-calling components that use its API. Ex.1003, ¶67; Ex.1005, [0241], [0243], [0247]. Accordingly, the only way the API-based functions of Abuan’s video conference module 1302 (“*function block*”) would have been accessed is through the stand-alone video conferencing client application (“*superblock application*”), and not through other applications on the mobile device or the mobile device’s operating system. Ex.1003, ¶¶67-69.

Thus, Abuan discloses or at least renders obvious “*the function block is accessible only to the superblock application on the electronic device.*”

[1.2] wherein the function block is operative when executed by a processor of the electronic device to:

Abuan’s video conference modules (*function block*) “are specified as a set of instructions recorded on a computer readable storage medium” of a mobile device (*electronic device*), and “executed by one or more processing unit(s).” Ex.1005,

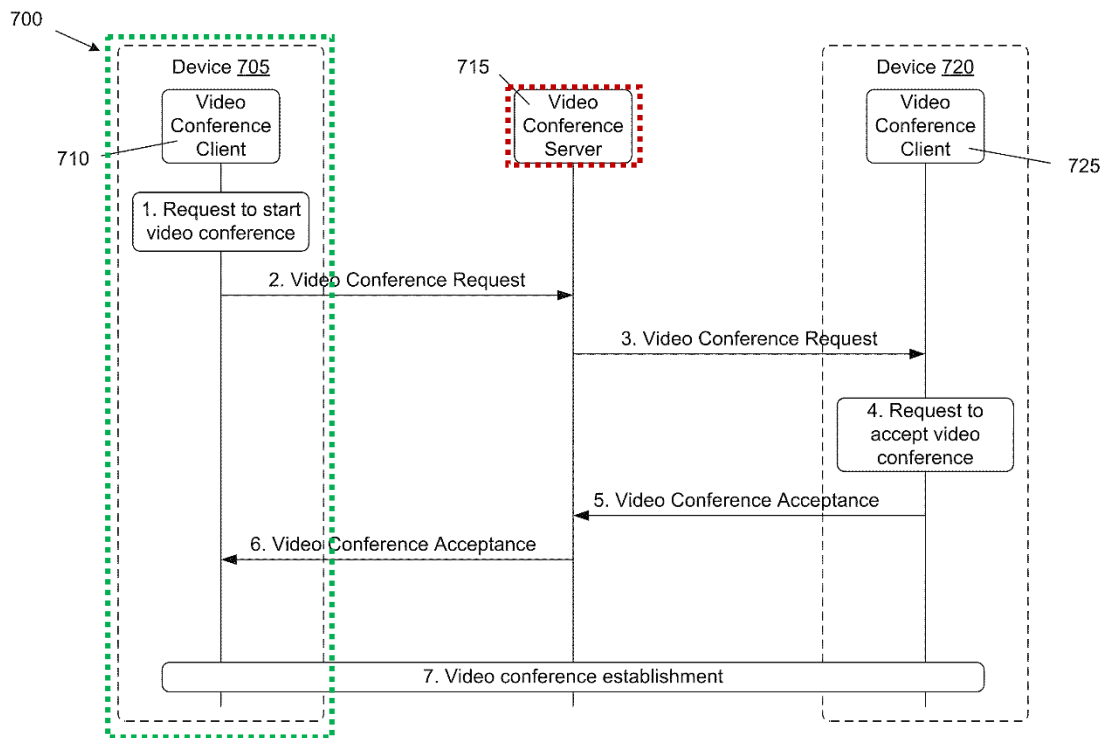
[0233], [0032]. Thus, Abuan's video conference modules (*the function block*) are

“operative when executed by a processor of the electronic device.” Ex.1003, ¶¶71-

72.

[1.3] establish a connection with an external device via a network interface to provide a service requested by the superblock application without switching context from the superblock application to another application on the electronic device.

“establish a connection with an external device”: Abuan's “video conference request messaging sequence 700” that initiates a videoconference in Fig. 7 shows a video conference client (e.g., client application 1365) sending a “Video Conference Request” message to a video conference server 715, intended for a second device 720. Ex.1005, [0108]-[0109]; Fig. 7.



Ex.1005, FIG. 7; Ex.1003, ¶74

The video conference server 715 is shown to be a separate device from the “client device 705” running the video conference client application. Ex.1005, Fig. 7. The video conference server 715 is “*an external device.*” Ex.1003, ¶¶74-76. The video conference client application 710 transmitting (e.g., video conference request) and receiving (e.g., video conference acceptance) from the server 715 teaches “*establish a connection with an external device.*” Ex.1005, [0111], [0113]; Ex.1003, ¶¶76-78. As explained above, Abuan discloses that the video conference clients 710, 725 rely on the functionality of the video conference module 1302 (e.g., networking manager 1314, video conference manager 1304) (function block)

to perform tasks like sending and responding to video conference requests.

“...via a network interface”: Abuan explains that the video conference clients 710 and 725 communicate with the server and each other via “a network interface **such as the network interface 650 described above.**” Ex.1005, [0109].

The “**network interface 650**” is used to send and receive data from networks “e.g., a local area network (LAN), a wireless local area network (WLAN), a wide area network (WAN), a network of networks, a code division multiple access (CDMA) network, a GSM network, etc.” Ex.1005, [0101]; Ex.1003, ¶79.

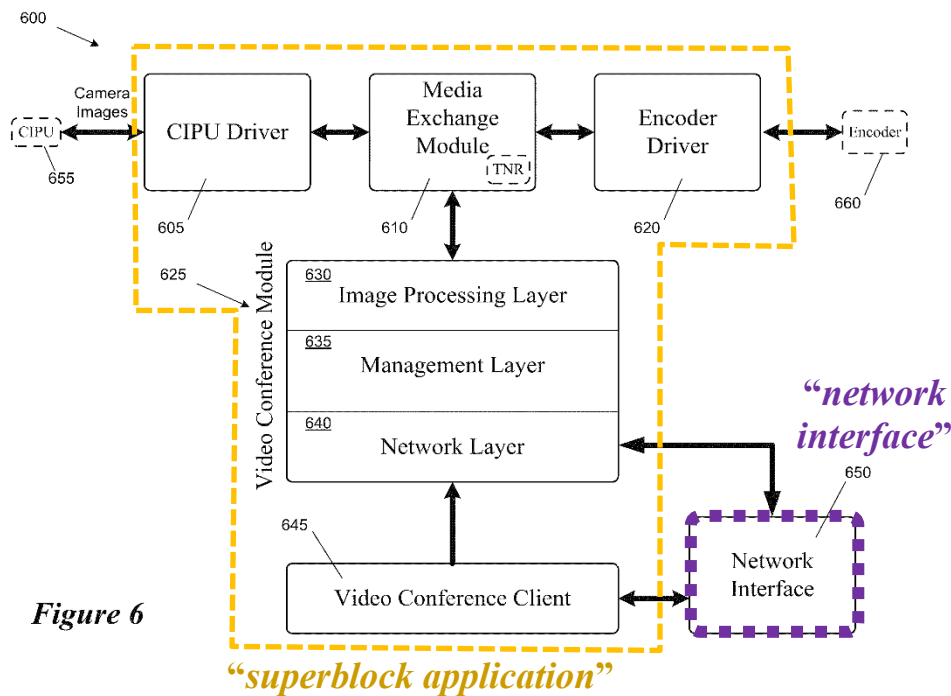


Figure 6

Ex.1005, FIG. 6 (annotated)

The video conference module (part of the *function block*) “send[s] data and receive data over a network (e.g., a cellular network, a local area network...the

Internet, etc.) through the network interface 650.” Ex.1005, [0106]. Abuan further explains that the network interface 650 is used for communications with the server 715:

Each of the video conference clients 710 and 725 of some embodiments communicates **with the video conference server 715** over a network (e.g., a cellular network, a local area network, a wireless network, a network of networks, the Internet etc.) **through a network interface** such as the network interface 650 described above.

Ex.1005, [0109].

Accordingly, the client 710 sends instructions to the video conference module, and the video conference module includes the functionality to carry out the instructed actions. Ex.1005, [0172]. Abuan teaches that the networking manager 1314—a submodule of video conference module 1302 (*function block*)—is responsible for establishing connections using the network interface 650.

Ex.1005, [0172], [0179]. Accordingly, Abuan discloses that the networking manager (*function block*) establishes the connection between the video conference client and the server (*external device*) “*via a network interface.*” Ex.1003, ¶¶80-83.

“*to provide a service requested by the superblock application*”: Abuan describes providing videoconferencing services (“*provide... a service*”), for example, by exchanging call setup messages with a videoconference server.

Ex.1005, [0108]-[0109] & Fig. 7. These services are provided by components within *the function block*, such as video conference module 1302, networking manager 1314, and network layer 640 and their submodules, such as video conference manager 1304. Ex.1005, [0186]-[0195]. Abuan explains that the client application “uses the video conferencing functions of the video conference module 1302.” Ex.1005, [0171]. Within video conference module 1302, a video conference manager 1304 “is responsible for initializing” various modules “when a video conference is starting, controlling the operating of the video conference module 1302 during the video conference, and ceasing the operation of some or all the other modules... when the video conference is ending.” Ex.1005, [0175]. The video conference module also includes a network layer 640 that “performs ... all of the networking functionalities for video conferencing.” Ex.1005, [0102]; *see id.* [0174] (“video conference module 1302 is the same as the video conference module 625 [containing network layer 640]”). Thus, Abuan describes how the various modules comprising *the function block* provide videoconferencing services (*a service*) requested by the client application (part of the “*superblock application*”). Ex.1003, ¶¶84-85.

“without switching context from the *superblock application* to another *application on the electronic device*”: Abuan describes the application employing the videoconferencing module and other modular components) as “a stand-alone

application.” Ex.1005, [0105]. A POSITA would have understood Abuan’s reference to a “stand-alone application” to mean that all the functionality of the application, including its module subcomponents (e.g., *the function block* functionality) is contained in a single, unitary application. *See* Ex.1012, 1:26-29 (“Traditionally, an application's source files are compiled in object modules and then linked together with other object modules, generically called libraries, to form a complete stand-alone application.”). Thus, it would have been obvious to a POSITA for the *establish*[ing], and *provid*[ing] functions performed by the *function block* would be performed within the execution context of Abuan’s client application and not some other application. Ex.1003, ¶¶86-89.

Additionally, Abuan describes in detail the use of application programming interfaces (APIs) to facilitate an application invoking functionality provided by a software library or other module. *See generally*, Ex.1005, [0236]-[0248]. Abuan does not reference making any context switch as part of the API-calling process. Thus, it would have been obvious to a POSITA from Abuan’s silence about context switching that there is no context switch performed when an API-calling component (such as Abuan’s client application) invokes a function or feature of an API-implementing component (such as Abuan’s videoconference module). Ex.1003, ¶¶86-87; *see also* ¶¶67-69. This is consistent with how a POSITA would have understood an application to execute; that is, a POSITA would have

understood that all the code for a stand-alone application would execute within the same memory context. Ex.1003, ¶88; Ex.1042, 317 (“We first need to make sure that each process has a separate memory space. To do this, we need the ability to determine the range of legal addresses that the process may access and to ensure that the process can access only these legal addresses.”), 321 (“When a routine needs to call another routine,... the relocatable linking loader is called to load the desired routine into memory and to update the program’s address tables to reflect this change...”).

Thus, Abuan renders obvious the video conference module 1302 operative to “*establish a connection with an external device via a network interface to provide a service requested by the superblock application without switching context from the superblock application to another application on the electronic device.*”

Ex.1003, ¶89.

[2.0] The non-transitory computer readable medium of claim 1 wherein the function block is further operative when executed by the processor to:

See [1.0], [1.2]; Ex.1003, ¶90.

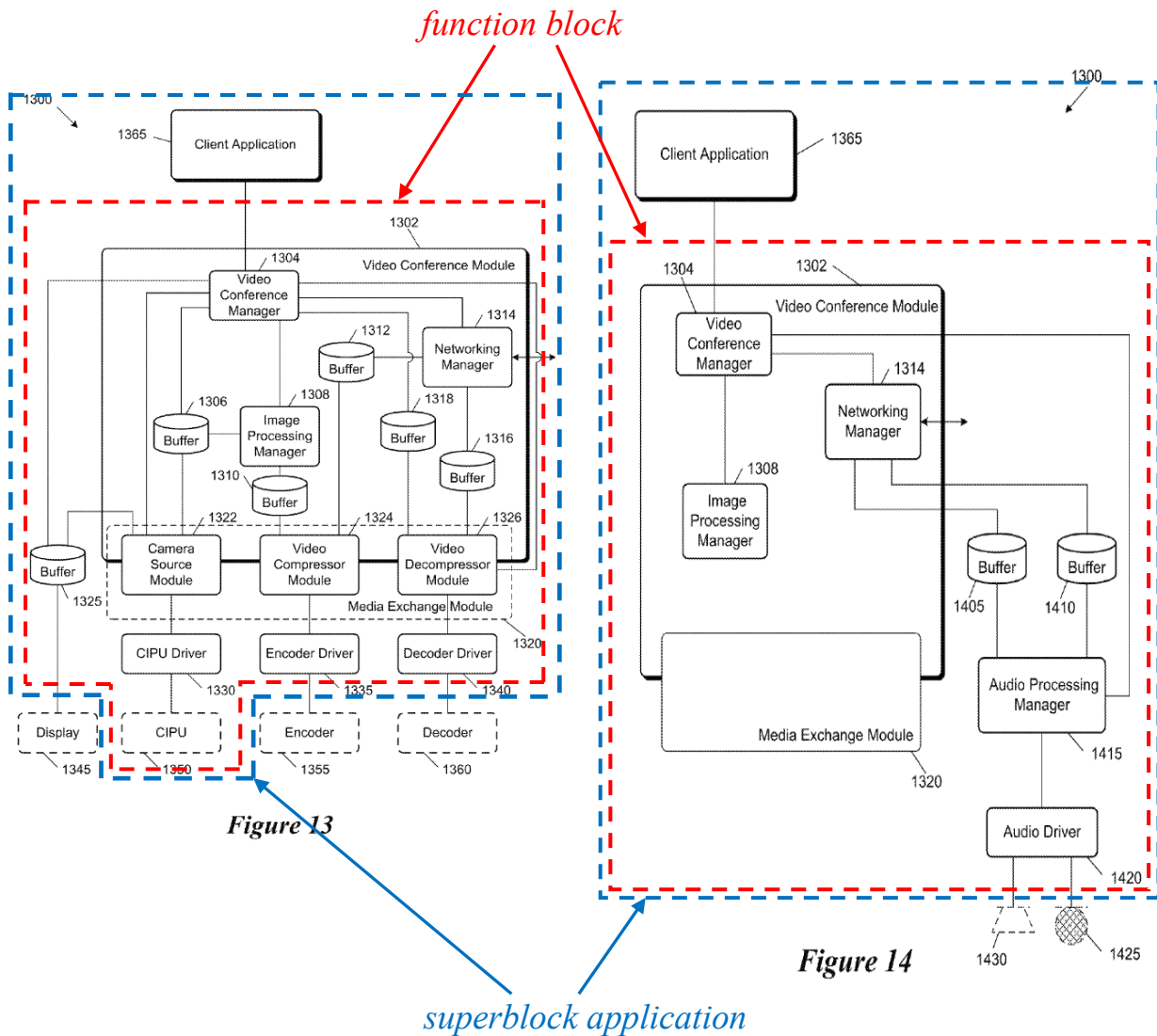
[2.1] receive from the superblock application a request for the service,

As explained above for limitation [1.0], the video conferencing and processing module 1300 (including the client application 1365 and the various modules) corresponds to the claimed “*superblock application.*” Abuan’s various modules and components (i.e., the video conferencing and processing module 1300

excluding the client application 1365) correspond to the claimed “*function block*.”

See, e.g., Ex.1005, [0056], [0099], [0170], [0183]; Ex.1003, ¶91.

The *superblock application* and *function block* are annotated in Abuan’s Figures 13 and 14 below.



Ex.1005, Figs. 13 and 14 (annotated).

“*function block* [is operative to] receive, from the *superblock application*, a request for the service”: Abuan describes how the client application 1365 “sends

instructions to the video conference module 1302 such as instructions to start a conference and end a conference,” such as an audio or video conference. Ex.1005, [0172], Abstract (“video conference”), [0185] (“multi-participant audio conference”). Abuan’s instruction to start or end a conference are each an example of a “*request for the service.*” Ex.1003, ¶92. Abuan explains that, based on a user’s selection of “a user interface (UI) item,” the video conference client 710 (corresponding to the video conference client 1365), “sends---a video conference request...to the video conference server 715.” Ex.1005, [0110]-[0111]. Abuan also describes the video conference module (part of the “*function block*”) *receiv[ing]* the instructions from the client application (part of the “*superblock application*”). Ex.1005, [0101]; Ex.1003, ¶92.

In one implementation of a “*request*” instruction sent from the client application to the video conference module 1302, Abuan explains that the client application makes API calls to API-implementing components of the video conference module to “access[] various video processing and encoding functionalities.” Ex.1005, [0237]; Ex.1003, ¶¶91-95.

Thus, Abuan discloses or at least renders obvious the video conference modules (*function block*) configured to “*receive from the superblock application a request for the service.*”

[2.2] wherein the service is a communication session, and

As explained above for limitation [2.1], the service requested from the user's selection includes a request to start a video conference session (a *communication session*). Ex.1005, [0110], [0111], [0172]. Thus, Abuan discloses this limitation. Ex.1003, ¶¶97-98.

[2.3] wherein the function block is configured to provide functionality for the communication session to the superblock application; and

Abuan explains that “client application 1365 [*the superblock application*]... uses the video conferencing functions [*functionality*] of the video conference module 1302 [*the function block*].” Ex.1005, [0171]. That functionality includes “image processing, video conference management, and networking.” Ex.1005, [0101]. For instance, the networking manager 1314 “manages network connections,” and the “video conference manager...control[s] the operation of the video conference module 1302 during the video conference.” Ex.1005, [0208], [0211], [0175]. The image processing manager 1308 processes and encodes the images obtained by the mobile device to be transmitted to the other mobile device in the video conference. Ex.1005, [0178]. Ex.1003, ¶99.

Thus, Abuan discloses or at least renders obvious the video conference modules being “*configured to provide functionality for the communication session to the superblock application*.” In that regard, a POSITA would have found it obvious for client application 1365 itself not to support the functionality provided by the video conference module 1302 because that would render the modules

superfluous and unnecessary. Ex.1003, ¶¶28-32, 100-101; *see* Ex.1032, 5:44-68.

Thus, Abuan discloses or at least renders obvious this limitation.

[2.4] notify the superblock application that the communication session is being requested.

As explained above for limitation [2.1], Abuan teaches that the client application provides a UI (e.g., Figs. 8-9) for the user to make selections and to request, accept, and end a video conference session. Ex.1003, ¶102; *see* Ex.1005, [0110] (“The video conference client 710... receives the request to start the video conference when the user of the device 705 selects a user interface (UI) item....”); *see also id.* [0112], [0115], [0187]. Further, Abuan explains that, when the video conference module receives instructions from the client application to start a video conference call, a video conference tool on the client application is activated to establish the audio/video conference call, and a hold stage or wait state indicator is displayed on the user interface of the client application 1365 (part of “*the superblock application*”) to indicate the activation and that the mobile device is waiting for a response from the other mobile device (to accept or reject the audio/video conference call). Ex.1005, [0108]-[0115]; Ex.1003, ¶¶102-103.

For example, Figure 8 of Abuan shown below shows the user interfaces of the client application displayed on a device 705 during the establishment of the audio/video conference call between device 705 and another device 720. Ex.1005, [0110]. The user interfaces belong to the client application on device 705. Ex.1003,

¶104. The user interfaces display a “smaller version of the video captured by the device,” “a still image that is stored on the device,” “a full screen display of a video being captured by the device,” “a ‘Preview’ notation at the bottom of this video,” etc., to indicate that the audio/video conference call is in the process of being established.

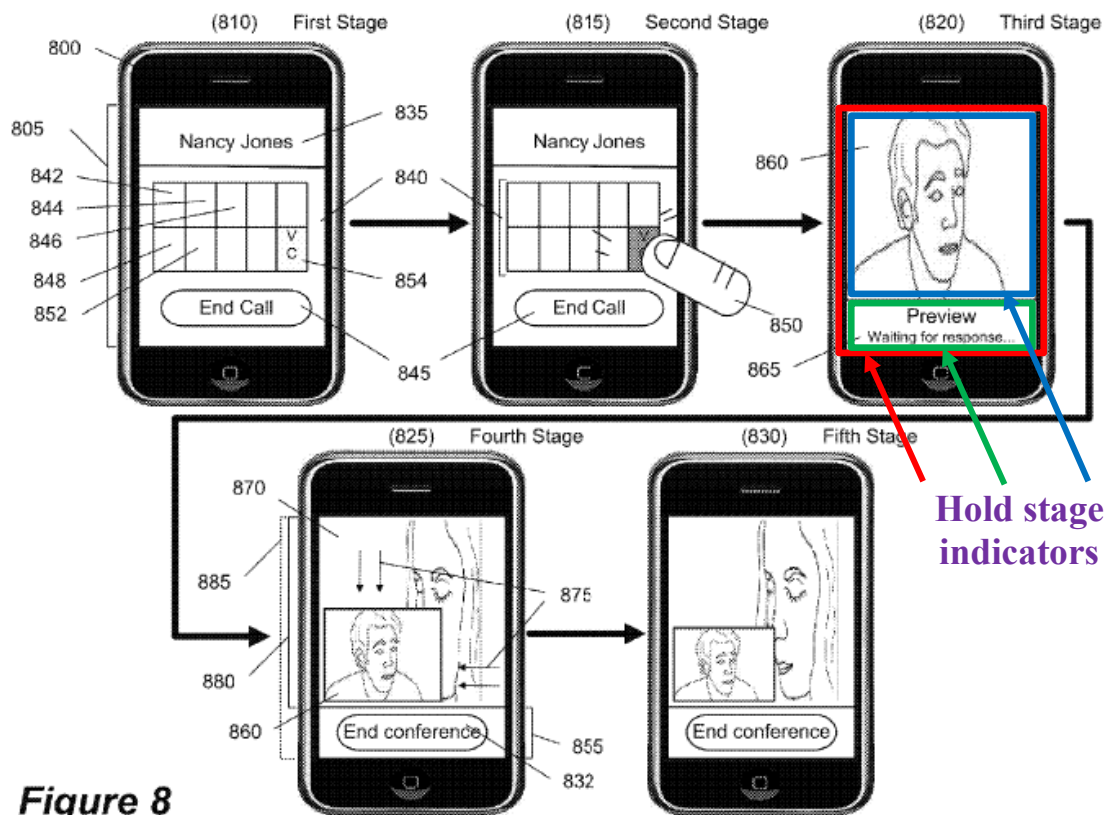


Figure 8

Ex.1005, Fig. 8 (annotated).

A POSITA would have found it obvious that *the function block*, in response to receiving the instructions from *the superblock application*, *notif[ie]d the superblock application* about the activation of the video conference tool (“*that the communication session is being requested*”) and the wait for the establishment of

the video conference call such that *the superblock application* displayed the hold stage indicator (e.g., as shown in Figure 8 above (third stage)). Ex.1003, ¶¶105-106.

First, because the video conference manager 1304 communicates with the client application to handle the client application's requests to establish video conferences as well as to cause the client application to update its user interfaces, the video conference manager 1304 (part of "*the function block*") would have been an obvious way to notify the client application (part of "*the superblock application*") about the status of the video conference request from the client application (i.e., about the wait for the establishment of the video conference call) so that the client application can update its user interface accordingly (e.g., by transitioning from second stage 815 to third stage 820). Ex.1003, ¶¶107-109.

Second, a user interface of a client application displaying a notification when a module, triggered by an event, informed the client application about the event was a routine and expected design in event-driven systems—common in mobile video conferencing. Ex.1003, ¶110. For example, a POSITA would have understood that the transition from second stage 815 to third stage 820 was triggered by an event, such as the user's selection of the Video Conference option 854 on the user interface of the client application 1365. Ex.1005, [0120] ("The third stage 820 displays the UI 805 after... the selection of the Video Conference

option 854.”). In response, a functional module, such as the video conference manager, informed the client application about the event, causing the client application to update its user interface to display a notification about the event, the hold stage indicator shown in Figure 8 above. Ex.1005, [0121]-[0123]. A POSITA would have understood that in event-driven systems like Abuan (*see* Ex.1005, [0140], [0243]), modules signal state transitions to the user interface via callbacks, queues, or observer notifications to inform users of backend. Ex.1041, 137 (defining “a callback” as “[a] function written as part of the application, associated with a specific widget resource, that is invoked as a result of a specific change of state associated with that widget). Ex.1003, ¶¶110-111.

[3.0] The non-transitory computer readable medium of claim 2

See [1.0], [1.2], [2.0]. Ex.1003, ¶112.

[3.1] wherein the steps of receiving, notifying, and establishing are performed without switching context from the superblock application to another application on the electronic device.

For the same reasons explained above for limitation [1.3], Abuan discloses or renders obvious the connection with the server and setting up the video conference being “*performed without switching context from the superblock application to another application on the electronic device.*” Indeed, Abuan does not reference making any context switch for any function calls performed when requesting, accepting, and establishing the video conference—including updating

the wait state indicator based on the *notif*[ication] provided to the client application 1365. Ex.1003, ¶¶112-113.

Thus, it would have at least been obvious to a POSITA from Abuan’s silence about context switching, and Abuan’s description of a “stand-alone application,” that there is no context switch performed during any of the “*receiving, notifying, and establishing*” steps discussed above. Ex.1003, ¶114.

Accordingly, Abuan renders this limitation obvious.

[4.0] The non-transitory computer readable medium of claim 2 wherein the function block is further operative when executed by the processor to

See [1.0], [1.2], [2.0]; Ex.1003, ¶¶116-117.

[4.1] determine whether the function block has a capability to perform any rendering needed by the communication session.

Abuan explains that the client hardware device includes common output devices such as “speaker 1430” and “a display” that play or show *render*[ed] audiovisual data for the user. Ex.1005, [0184], [0252]. Those devices rely on respective drivers, such as the audio driver 1420 and the decoder driver 1340, to decode and process the incoming data so that it can be output to their respective audiovisual peripherals. Ex.1003, ¶¶118, 121. For instance, the “audio processing manger 1415...retrieves audio data from the buffer 1410 and decodes the audio data, which is then output through the audio driver 1420 to the speaker 1430.” Ex.1005, Fig. 14, [0183]-[0184]. Similarly, the decoder driver 1340 has

functionality to decode images received during the video conference, and provides decoded images to the video decompressor module 1326. Ex.1005, [0181]-[0182]. Thus, audio driver 1420 and decoder driver 1340 have capabilities to *render*[] the audio and video data received in the video conferencing session. Ex.1003, ¶¶118-123.

A POSTIA would have appreciated that the hardware components responsible for decoding the audiovisual data would have had capabilities (e.g., supported formats) needed for *rendering* incoming video/audio streams. Ex.1003, ¶123. It would have been obvious for the audio processing manager 1415 and/or the decoder driver 1340 to query their respective drivers to determine their rendering capabilities. Ex.1003, ¶¶119, 123. In that regard, Abuan explains that one implementation of a “device driver” is an “API-implementing component.” Ex.1005, [0246]. Abuan further explains that API calls “report to an application the capabilities or state of a hardware device, including...**output capabilities.**” Ex.1005, [0243].

A POSITA would have been familiar with device drivers for output devices and would have recognized that they commonly included APIs for reporting on an output device’s capabilities. Material Fact #3; Ex.1003, ¶121; *see* Ex.1033, 6:30-32 (using “software (e.g., the device drivers) to determine the capabilities” of hardware resources); Ex.1034, 11:8-26 (describing device drivers determining a

“display device’s maximum resolution”); Ex.1035, Abstract (“determine capabilities of graphics hardware from a device driver”), [0004]; Ex.1036, [0033].

It would have been obvious to a POSITA for Abuan’s video conference modules (e.g., audio manager 1415, decoder driver 1340) (*function block*) to determine their output capabilities, including audio and video encoding formats that are supported for decoding. Ex.1003, ¶¶121-123.

Querying the output capabilities of the mobile device’s hardware is consistent with the ’046 Patent’s disclosure of determining whether the function block has rendering capabilities for a video conference. Ex.1003, ¶124; *see* Ex.1001, 11:26-52 (describing checking for “an available speaker output” or “whether there is an available screen”).

Thus, Abuan renders obvious this limitation.

[6.0] The non-transitory computer readable medium of claim 1 wherein the function block is further operative when executed by the processor to

See [1.0], [1.2]; Ex.1003, ¶¶126-127.

[6.1] provide at least one configuration parameter for the function block,

Abuan describes various configurable settings (*configuration parameter[s]*) used for a video conference, including “a profile setting for configuring a camera” to use “high resolution video,” “low resolution video,” “still images,” or various power levels. Ex.1005, [0068], [0058]. Additionally, Abuan discloses that the user can interact with the client application 1365 to change settings, such as “as

“switching cameras,” “focus adjustment, exposure adjustment,” or a change to network settings. Ex.1005, [0191]-[0193]. Abuan explains that the camera is controlled via a “sensor module” or “camera source module,” either of which is part of *the function block*. Ex.1005, [0057], [0180]. Each of these settings is an example of a “*configuration parameter*” that is provided to the *function block* and that controls aspects of the video conference. Ex.1003, ¶¶128-129.

Abuan discloses that API calls are used to “pass[] data and control information using **parameters**,” and that such API calls may be sent from one API-implementing component to another API-implementing component. Ex.1005, [0241], [0240] (“[T]he API-implementing component may itself call one or more other components via an underlying API and thus be both an API-calling component and an API-implementing component.”)

Thus, Abuan’s disclosure of the video conference module receiving instructions to change configuration settings, and its disclosure of passing parameters from an API-calling component (or API-implementing component) of the video conference module 1302 (part of the *function block*) to another API-implementing component, discloses or renders obvious “*provide at least one configuration parameter for the function block*.” Ex.1003, ¶¶130-131.

[6.2] wherein the configuration parameter enables the function block to be configured after the function block is distributed as part of the superblock application.

As explained above for limitation [6.1], Abuan's camera profile settings teach or render obvious the "*configuration parameter*." The camera profile settings may be selected based on a running application: "[I]f the user selects a video conferencing application, a profile that configures both cameras to capture video is automatically selected." Ex.1005, [0068]. Thus, it would have been obvious to a POSITA for the sensor module or camera source module to receive, as a *configuration parameter*, information about how the client application intends to use the camera. Ex.1003, ¶132. This configuration step would be performed as part of running the client application, since the configuration occurs in response to the user's actions. Thus, the camera's configuration is "*configured after the function block is distributed as part of the superblock application*." Ex.1003, ¶¶132-133; *see also supra*, [1.3] (the client application is a "stand-alone application" and thus incorporates the *function block* modules).

[9.0] The non-transitory computer readable medium of claim 1 wherein the function block is further operative when executed by the processor to

See [1.0], [1.2]; Ex.1003, ¶¶135-136.

[9.1] receive a request for the service from the superblock application as an application programming interface (API) call from the superblock application to the function block.

As discussed at [2.1], Abuan's instruction to start or end a conference are each an example of a "*request for the service*." As also discussed at [1.3] and [2.1], Abuan contemplates using an API between a calling component (e.g., Abuan's

client application, which is a *superblock application*) and an implementing component (e.g., Abuan’s videoconferencing module, which is part of a *function block*). Ex.1003, ¶137.

Abuan further explains that an API call is used “to initiate performance of an operation.” Ex.1005, [0243]. Starting or ending a videoconference is an operation, so it would have been obvious to a POSITA for Abuan’s client application to initiate such operations by calling one or more functions or methods defined in the API of the videoconferencing module. Ex.1003, ¶138. Such use of an API would have also been obvious because the use of APIs was common in the prior art. Material Fact #2; *see* Ex.1003, ¶68.

Thus, Abuan discloses or at least renders obvious the user’s initiated request to start a video conference being received “*as an application programming interface (API) call from the superblock application to the function block.*” Ex.1003, ¶139.

[10.0] The non-transitory computer readable medium of claim 1 wherein the function block is further operative when executed by the processor to

See [1.0], [1.2]; Ex.1003, ¶¶140-141.

[10.1] determine whether the service is available on the external device.

Abuan illustrates in Figure 7 a process for establishing a videoconference connection (part of “*the service*”) to a videoconference server (“*the external device*”). Within Abuan’s videoconference module 1302 (part of “*the function*”

block”), a “session negotiating manager 1705 is responsible for establishing connections between the dual camera mobile device and one or more remote devices participating in the video conference.” Ex.1005, [0211]. The session manager employs communication protocols such as “transmission control protocol (TCP),” hypertext transfer protocol (HTTP), and “session initiation protocol (SIP).” Ex.1005, [0210].

A POSITA would have been familiar with TCP, HTTP, and SIP, including that HTTP and SIP use TCP. Ex.1003, ¶142; Ex.1050, §1.5; Ex.1051, 11 (“HTTP communication usually takes place over TCP/IP connections.”); Ex.1052, 142 (“All SIP elements MUST implement UDP and TCP.”). Initiating a TCP connection involves the client sending a synchronization message (SYN) to a server and waiting to receive an acknowledgement message (ACK) in return. Ex.1044, [0023]. Thus, before sending a videoconference request to the videoconference server, Abuan’s session negotiating manager would first send a synchronization message to initiate a TCP connection to the videoconference server 715. It would have been obvious to a POSITA that if the videoconference server fails to return an acknowledgement message, the videoconference server is not available. Conversely, if the videoconference server returns an acknowledgement message, the videoconference server is available. Thus, by initiating a TCP connection to the videoconference server (sending a SYN message

and receiving an ACK in return), Abuan's session negotiating manager renders obvious determining whether the video conference functionalities ("*the service*") are "*available*" on the server ("*on the external device*") as claimed. Ex.1003, ¶¶143-146.

3. Independent Claim 11 and dependent claims

Claim 11 recites substantially the same functionality as claim 1 in the form of a method claim, except for a few differences discussed below. As discussed above for claim 1, Abuan describes how a video conference module 1302 and related other modules perform a variety of "video conferencing functions," including "video processing and encoding functionalities" and "audio processing operations." Ex.1005, [0171], [0237], [0183]-[0184]. These functions and operations render obvious perform "*a method.*" Ex.1003, ¶¶147-149.

Additionally, where claim 1 recites an "*establish a connection with an external device...to provide a service,*" claim 11 recites "*accessing a resource...to provide a service.*" This difference does not affect change the analysis for claim 11, as Abuan's video conference server 715 likewise corresponds to the "*resource,*" and Abuan discloses that part of providing video conference services involves communicating with (*accessing*) the video conference server 715. Ex.1003, ¶¶150-151; see [13.1] ("*wherein **the resource** is **external** to the electronic device*").

Second, claim 11 recites that "*the service is not otherwise supported by the*

superblock application.” This limitation is addressed separately below. For the remaining limitations, the analysis is identical to that for claim 1 above.

[11.0] A method implemented in a function block associated with an electronic device having a network interface, comprising:

See [1.0] (“a computer program comprising instructions that include a function block”), [1.1] (“function block is accessible only to the superblock application on an electronic device”), [1.3] (“[function block is operable to] establish a connection...via a network interface”). Ex.1003, ¶¶147-149.

[11.1] accessing a resource via the network interface to provide a service to a superblock application on the electronic device,

See [1.3]; Ex1003, ¶¶150-151.

[11.2] wherein the service is not otherwise supported by the superblock application, and

As shown in Fig. 7 and explained above for limitation [1.3], Abuan’s video conference requests, acceptances, and other messages depend on communication with the video conference server to route and forward those messages to establish the video conference session. *See* Ex.1005, [0109]. Thus, a POSITA would have understood that Abuan’s video conferencing functionality (“*the service*”) is *not otherwise supported* by the video conference clients (“*superblock application[s]*”) except through communication with the video conference server 715 via the *network interface*. Ex.1003, ¶¶153-154.

Additionally, as explained above for limitation [1.3], Abuan discloses or

renders obvious the video conference client relying on the software modules of the video conference module 1302, compiled together with the video conference client, to conduct a video conference. *See* Ex.1005, [0179]. It would have been obvious for client application 1365 to *not otherwise support*[] the functionality provided by the video conference module 1320 because duplicating the program code providing that functionality would be superfluous and needlessly complex. Ex.1003, ¶155; *see* Ex.1032, 5:44-68.

Accordingly, Abuan discloses or at least renders obvious for video conferencing (*the service*) to “*not [be] otherwise supported by the superblock application*” except as specified in limitation [11.1]. Ex.1003, ¶156.

[11.3] wherein the function block is accessible only to the superblock application on the electronic device; and

See [1.1]; Ex.1003, ¶157.

[11.4] providing the service to the superblock application without switching context from the superblock application to another application on the electronic device.

See [1.3]; Ex.1003, ¶¶158-160.

[14.0] The method of claim 11, further comprising

See [11.0]. Ex.1003, ¶161.

[14.1] operating as a bridge between the superblock application and the resource.

As discussed at [1.0] and [1.2], Abuan illustrates in Fig. 13 various “*function*

block” modules that provide functionality to the client application, including networking manager 1314. Networking manager 1314 “manages network connections,” including “establishing connections... [with] one or more remote devices.” Ex.1005, [0208], [0211]. Since networking manager 1314 is the only component in Figure 13 described as having network connectivity, it would have been obvious for networking manager 1314 (part of the “*function block*”) to act “*as a bridge*” by facilitating communications between the client application (part of “*the superblock application*”) and videoconference server 715 (“*the resource*”) Ex.1003, ¶162.

[15.0] The method of claim 11, further comprising

See [11.0]. Ex.1003, ¶163.

[15.1] initializing the function block when the superblock application is loaded and executed.

Abuan describes a process 1500 that begins with “receiving (at 1505) instructions to start a video conference.” Ex.1005, [0187]. For example, the video conference may be started by the user interacting with the user interface of the client application (part of the “*superblock application*”) shown in Figures 8-10. *See, e.g.*, Ex.1005, [0110]. The user’s interaction, in turn, causes the client application to start initializing the modules required for a videoconference. Ex.1005, [0187] (“instructions to start a video conference... are received... from a user through a user interface... and forwarded to the video conference manager

1304 by the client application 1365”). Ex.1003, ¶¶164-165.

Abuan describes initializing the videoconference manager 1304 and other modules in response to the request to start a videoconference:

Next, the process 1500 **initializes (at 1510) a first module** that interacts with the video conference manager 1304. The modules of some embodiments that interact with the video conference manager 1304 include the CIPU 1350, the image processing manager 1308, the audio processing manager 1415, and the networking manager 1314.

Ex.1005, [0188]; *see also id.* [0189]-[0190] (describing how initialization continues until “all of the required modules have been initialized”).

Since the module initialization occurs in response to user’s interaction with the client application user interface, it would have been obvious to a POSITA for initialization to occur when the client application “*is loaded and executed.*”

Ex.1003, ¶¶166-167. For example, Abuan describes how an application is “read into memory for processing by a processor.” Ex.1005, [0234]; *see also id.* [0233] (“applications are implemented as software processes that are specified as a set of instructions... executed by one or more processing unit(s)”). Ex.1003, ¶167.

[16.0] The method of claim 11, further comprising

See [11.0]. Ex.1003, ¶168.

[16.1] providing at least one configuration parameter for the function block,

See [6.1]; Ex.1003, ¶¶168-169.

[16.2] wherein the configuration parameter enables the function block to be configured after the function block is distributed as part of the superblock application.

See [6.2] ; Ex.1003, ¶170.

[17.0] The method of claim 11, further comprising

See [11.0]. Ex.1003, ¶171.

[17.1] determining whether the function block is capable of performing any rendering needed by the service.

See [4.1]. *See also*, [2.2] (explaining that “*the service is a communication session*”); Ex.1003, ¶172.

[19.0] The method of claim 11, further comprising

See [11.0]. Ex.1003, ¶173.

[19.1] receiving a request for the service from the superblock application, wherein the request is received as an application programming interface (API) call from the superblock application to the function block.

See [2.1], [9.1]; Ex.1003, ¶174.

[20.0] The method of claim 11, further comprising

See [11.0]. Ex.1003, ¶175.

[20.1] determining whether the resource is available.

See [10.1]; Ex.1003, ¶¶176-177.

C. Ground 2

1. Ludwig

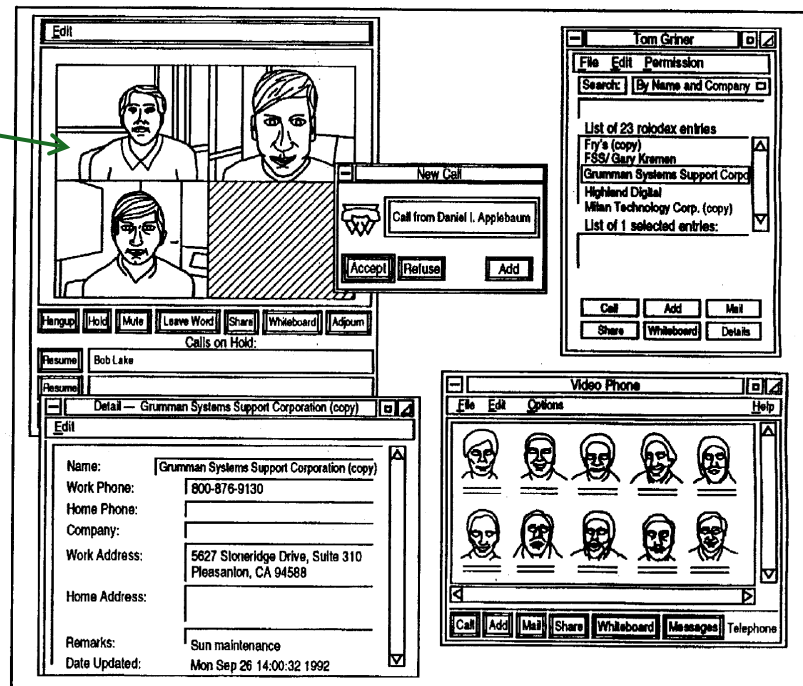
Ludwig describes a multimedia collaboration system that integrates

real-time audio/video teleconferencing with data conferencing to replicate face-to-face meetings across distributed networks and heterogeneous platforms.

Ex.1008, Abstract. Beyond high-quality, multi-party video and audio, the system emphasizes advanced collaboration tools, including “snapshot” or screen sharing of windows or regions, application sharing with shared control, and a shared whiteboard that supports telepointing and rich, multi-user annotations in real time.

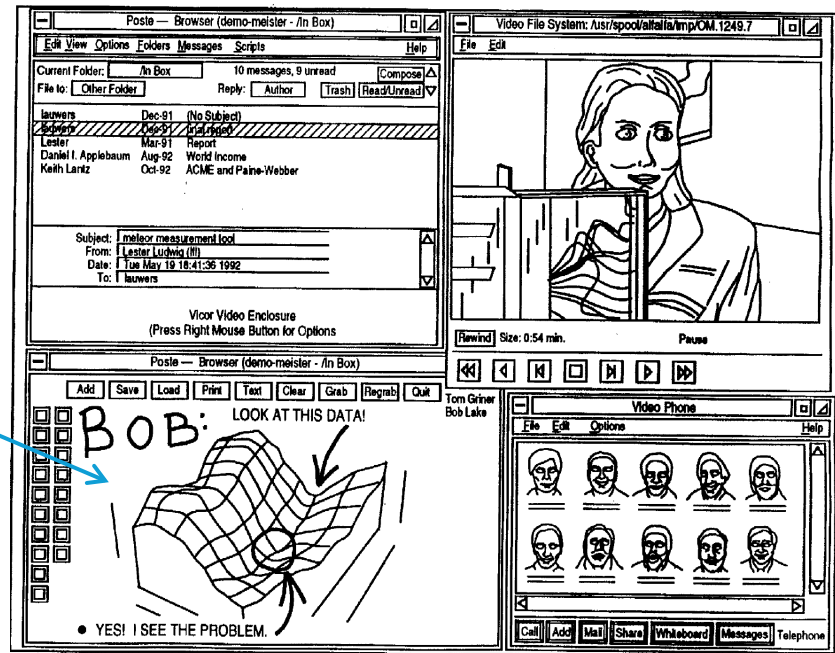
Ex.1008, 2:57-3:12. Ludwig illustrates exemplary mid-meeting screens in Figures 2A and 2B, including “full-motion video of three conference participants” and “data shared and annotated by those conferees.” Ex.1008, 6:53-57; Ex.1003, ¶¶178-179.

full-motion video of
three conference
participants



Ex.1008, Fig. 2A (annotated).

Data shared and annotated by conference participants



Ex.1008, Fig. 2B (annotated).

Ludwig is in the same field of endeavor as the '046 patent, namely, the “manner in which functionality is accessed in certain environments, such as mobile device environments.” Ex.1001, 1:66-67; *see* Ex.1008, 38:56-39:32 (describing a videoconference participant using a “laptop in the field in Central Mexico” equipped with “two wireless modems”).² Ludwig and the '046 patent also both generally relate to implementing functionality via software. *See* Ex.1001, Abstract

² Those of skill in the art in the 1990s (when Ludwig was filed and published) considered a laptop to be a mobile device. *See* Ex.1001, 3:19, 3:26-28; Ex.1045, 1:37 (“mobile devices, such as notebook and laptop computers”); Ex.1046, 4:19-24.

(“providing additional functionality to existing software”); Ex.1008, Fig. 20, 18:34-20:37 (describing “workstation software” and “server software” for multimedia collaboration). Ludwig and the ’046 patent both describe at length features for facilitating videoconferencing. Ex.1001, Abstract, 4:43-51 (purpose of a function block is to provide functions including “audio, video,... conferencing, meetings, and/or other functions”); Ex.1008, 2:66-3:7 (“As used herein, desktop teleconferencing includes real-time audio and/or video teleconferencing...”); Ex.1003, ¶180.

Ludwig is also reasonably pertinent to a problem allegedly addressed by the ’046 patent. The ’046 patent describes purported problems associated with using multiple applications on a mobile device, potentially requiring a user to “switch[] back and forth between the video window of [a] call and [a] superblock application.” Ex.1001, 4:22-26. The ’046 patent’s proposed solution is to provide functionality—such as communication functions—to a superblock application. Ex.1001, 4:43-51. Ludwig similarly describes how communication functions are centrally coordinated and provided via a “Collaboration Initiator”: “All collaborative functions can be accessed through this module.” Ex.1008, 18:53-55. Ludwig illustrates in Fig. 20 that the Collaboration Initiator is integrated with other data conferencing modules, such as videophone 169 and application sharing 166. The Collaboration Initiator and its integrated modules are shown separated from

“other applications 170,” indicating that Ludwig’s modules provide their communication functions to the Collaboration Initiator directly (not via the “interapplication protocols” used to communicate with other applications 170). Ex.1008, 18:44-52; Ex.1003, ¶181.

2. Reasons to combine Abuan and Ludwig

A POSITA would have been motivated to combine the teachings of Ludwig with Abuan for multiple reasons. Ex.1003, ¶¶191-193.

Abuan generally describes technology for facilitating videoconference communications and suggests the possibility of also sharing “other types of content.” Ex.1005, Abstract, [0037]. Ludwig explains that traditional videoconferences are “limited in the richness of the data that can be exchanged.” Ex.1008, 2:9-10. To address these limitations, Ludwig describes a “multimedia collaboration system that greatly facilitates distributed collaboration, in part by replicating the benefits of face-to-face collaboration.” Ex.1008, 2:59-62. Ludwig’s improved system adds features not found in traditional videoconferencing, such as “sharing of ‘snapshots’ of selected regions of the user’s screen,” “application sharing,” and a “shared “whiteboard.” Ex.1008, 3:1-6. The benefits of Ludwig’s multimedia collaboration system are described with a detailed scenario involving how users at a variety of remote locations access an “in-house corporate ‘expert’” regarding trading of financial instruments. Ex.1008, 35:35-41:37; Ex.1003, ¶191.

Notably, all the “collaborative activities in this scenario [take] place during a relatively short portion of the expert’s day (e.g., less than an hour of cumulative time) while the Expert remained in his office.” Ex.1008, 41:27-30; Ex.1003, ¶191.

A POSITA would have found it obvious to incorporate Ludwig’s multimedia collaboration features in Abuan’s videoconferencing system. Ex.1003, ¶¶191-193.

The further combination with Ludwig is simply the integration of prior art elements (e.g., videoconferencing, screen sharing, collaborative whiteboarding) according to known methods (implemented using ordinary software programming techniques within the POSITA skill set) to yield predictable results (adding screen-sharing and whiteboarding features to Abuan’s videoconferencing clients).

Ex.1003, ¶192. Alternatively, the combination is merely the use of known techniques (e.g., screen sharing, whiteboarding) to improve similar devices (Abuan’s videoconference clients and server) in the same way (expanding the available collaboration tools made available to users). Ex.1003, ¶¶192-193.

Additional reasons to combine are provided in the detailed analysis below.

3. Dependent claims 5 and 18

[5.0] The non-transitory computer readable medium of claim 1 wherein the function block is further operative when executed by the processor to

See [1.0], [1.2]. Ex.1003, ¶¶182-183.

[5.1] enable a user of the electronic device to collaborate on at least one of a document, a file, or an application with a user on another electronic device.

In addition to microphone-sourced audio and camera-sourced video communications, Abuan describes sending additional forms of content, such as “(1) files and other content stored on the [mobile] device, (2) the screen display of the device (i.e., the content that is displayed on the device’s screen), [or] (3) content received from another device....” Ex.1005, [0037].

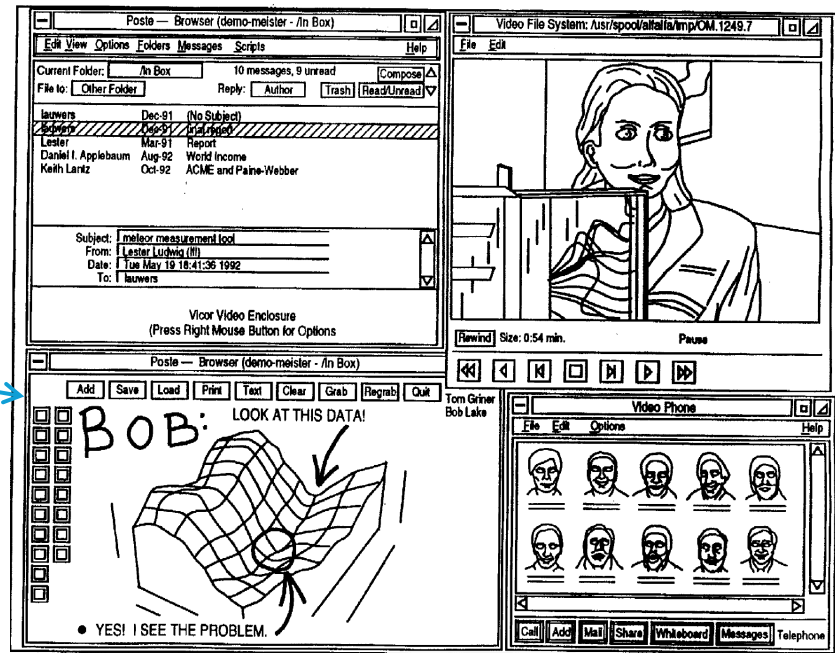
Ludwig provides further explanation and description of improving ordinary videoconferencing with additional collaboration features:

In accordance with the present invention, computer hardware, software and communications technologies are combined in novel ways to produce a multimedia collaboration system that greatly facilitates distributed collaboration, in part by replicating the benefits of face-to-face collaboration.

Ex.1008, 2:57-62.

Ludwig’s “multimedia collaboration” techniques allow a user to share their computer display with other conference participants, and allow a remote participant to control their computer. Ex.1008, 9:12-18. Users can also “point at or annotate the shared data.” Ex.1008, 9:22-23; *see also id.*, 9:23-29, Fig. 2B. Ludwig illustrates, for example, that multiple users have collaborated on a graph document, with one user writing “BOB: LOOK A THIS DATA!” and another user responding “YES! I SEE THE PROBLEM”:

Conference participants collaborating on a file or document



Ex.1008, Fig. 2B (annotated).

Ludwig describes and illustrates additional examples of users collaborating via shared documents and applications. Ex.1008, 36:49-37:6, 38:34-40, Figs. 36-37, 40; Ex.1003, ¶¶185-190.

It would have been obvious to improve Abuan's videoconferencing services to also include Ludwig's collaboration features, such as screen sharing and screen annotating, because Abuan expressly suggests incorporating such features. Ex.1003, ¶¶191-192; Ex.1005, [0037]. Ludwig also touts these features' advantages over traditional videoconferencing systems that lack them. *See, e.g.*, Ex.1008, 2:11-33, 3:53-59. In the combined system of Abuan-Ludwig, a user's request to start a videoconference would initiate a "data conference" as described by Ludwig, allowing users to not only see and hear one another, but also to use

Ludwig's collaboration tools such as screen sharing and screen annotating.

Ex.1003, ¶¶191-192; Ex.1008, 3:1-6. Thus, Abuan and Ludwig render obvious
“enable a user of the device to collaborate on at least one of a document, a file, or
an application with a user on another electronic device.” Ex.1003, ¶194.

Ludwig, like Abuan, describes a modular approach to software development, including a Collaboration Initiator component through which “All collaborative functions can be accessed.” Ex.1008, 18:34-20:37, Fig. 20; Ex.1003, ¶¶191-193. Accordingly, it would have been obvious for Ludwig's collaboration features to be implemented in Abuan by modifying or adding to the modules in Abuan's video conference module (“*the function block*”). Ex.1003, ¶191. Thus, the prior art renders obvious “*the function block*” being operable to “enable” the collaboration, as claimed. Ex.1003, ¶194.

[18.0] The method of claim 11, further comprising

See [11.0]. Ex.1003, ¶195, §VIII.B.3.a.

[18.1] enabling a user of the electronic device to collaborate on at least one of a document, a file, or an application with a user on another electronic device.

See [5.1]. Ex.1003, ¶195.

D. Ground 3

1. Lawson

Lawson describes a cloud-based communication platform that mediates incoming and outgoing voice, video, and messaging connections for client

applications. Ex.1007, [0027]-[0031], [0042]-[0047], [0055]-[0057]. Lawson describes a layered security model in which a policy engine authenticates client communications using authorization tokens and enforces permissions before permitting connection establishment. Ex.1007, [0028], [0033]-[0035], [0038]-[0039], [0048]; Ex.1003, ¶¶196-197.

Lawson is analogous art to the '046 patent. Lawson is in the same field of endeavor as the '046 patent, namely, the “manner in which functionality is accessed in certain environments, such as mobile device environments.” Ex.1001, 1:66-67; Ex.1007, [0046]-[0048] (describing a “functions to control the general functionality of the mobile device 14, including but not limited to its communication capabilities”). Lawson and the '046 patent also both generally relate to implementing functionality via software. *See* Ex.1001, Abstract (“providing additional functionality to existing software”); Ex.1007, [0060] (“enable software applications... on a mobile device to interface with a telephony or communication platform”). Lawson and the '046 patent both describe features for facilitating audio and video communications. Ex.1001, Abstract, 4:52-58; Ex.1007, [0034], [0048] (“communication session that supports communications selected from audio, video, chat, and/or other suitable communication mediums”), [0057]; Ex.1003, ¶198.

Lawson is also reasonably pertinent to a purported problem allegedly

addressed by the '046 patent, namely, the need to provide functionality—such as communication functions—to a superblock application. Ex.1001, 4:43-55. Lawson illustrates in Fig. 17 that a mobile device “include[s] a series of modules relating to its communication function.” Ex.1007, [0046]. These modules include an “audio engine 42” used “to control a majority of voice communication functions, including interfacing with any device hardware 44 such as speakers and microphones.” Ex.1007, [0046]. The audio engine will also “initiate and control communication sessions including voice communication sessions using Internet protocol.” Ex.1007, [0046]. The mobile device further includes a “cloud interface module” that is “configured within a browser function of the mobile device.” Ex.1007, [0047]. The cloud interface module is used by “untrusted client applications to prove that they are authorized to perform particular actions.” Ex.1007, [0048]; Ex.1003, ¶199.

2. Reasons to combine Abuan and Lawson

A POSITA would have been motivated to combine the teachings of Lawson with Abuan for multiple reasons. Ex.1003, ¶¶206-210.

Abuan generally describes client applications for facilitating videoconference communications. Ex.1005, Abstract. Lawson describes techniques “for communicating with a client application.” Ex.1007, Abstract. Lawson describes the benefits of employing “authentication and/or authorization security

measures that function to secure the communication channels,” such as thwarting attacks from “malicious parties.” Ex.1007, [0028]. Lawson’s techniques are imminently “adapted to function for any suitable type of communication,” including “video calls.” Ex.1007, [0028]; Ex.1003, ¶207.

A POSITA would have found it obvious to incorporate Lawson’s authentication and authorization security measures in the videoconferencing system of Abuan. The combination with Lawson is simply the integration of prior art elements (e.g., authentication, authorization checks, and videoconferencing) according to known methods (implemented using ordinary software programming techniques within the POSITA skill set) to yield predictable results (restricting access to Abuan’s videoconferencing system to authorized users). Ex.1003, ¶¶207-209. Alternatively, the combination is merely the use of known techniques (e.g., logging into a server) to improve similar devices (Abuan’s videoconference clients and server) in the same way (“prevent[ing] malicious parties from posing” as someone else). Ex.1007, [0028]; Ex.1003, ¶¶207-210.

Additional reasons to combine are provided in the detailed analysis below.

3. Dependent claims 7 & 12

[7.0] The non-transitory computer readable medium of claim 1 wherein the function block is further operative when executed by the processor to

See [1.0], [1.2]; *see also* Ex.1003, ¶200.

[7.1] determine that the superblock application is authorized to access

functionality provided by the function block for the service.

Abuan illustrates in Figure 7 a “video conference request messaging sequence 700” that initiates a videoconference. When a video conference client (e.g., client application 1365) sends a request to start a videoconference, a “Video Conference Request” message is sent to a video conference server 715. Ex.1005, [0108]-[0109]; Fig. 7. Thus, Abuan’s Fig. 7 process assumes that a user is already logged into, and authorized to use, the video conference server.

Like Abuan, Lawson describes a cloud-based communication system to support communication sessions between an initiating client device and a destination. Ex.1007, Abstract. The communication session can include “voice/audio, video, screen sharing,” or other suitable media. Ex.1007, [0057]; Ex.1003, ¶¶202-203.

Lawson describes using “security... management tools,” such as “an authorization token, which can be defined as an electronic listing of possible privileges for its bearer.” Ex.1007, [0048]. For example, authorization tokens are used to control “a user’s abilities to make and/or receive calls” or to request “communications selected from audio, video, chat, and/or other suitable communication mediums.” Ex.1007, [0048]; Ex.1003, ¶¶203-204.

Lawson explains that an authorization token is “verified” by checking its value at a “communication platform,” such as “servers and/or services configured

for trafficking and connection communications between [] users.” Ex.1007, [0051].

A POSITA would have found it obvious to incorporate Lawson’s authorization-checking concepts into the system Abuan for multiple reasons, including to prevent unauthorized users from wasting resources and bandwidth of the videoconference server. Ex.1003, ¶¶206-210.

In the combination, Lawson’s authorization token would be used to authorize Abuan’s client application (and the user of that application) to “make and/or receive [videoconference] calls.” Ex.1007, [0048]. Abuan’s client application would include the authorization token with its instructions to the videoconference module to start a videoconference. The videoconference module would then verify the received authorization token by communicating with the videoconference server (which would incorporate the verification techniques Lawson describes for a communication platform such as a server). This server-supported verification process would be used to determine whether the client application is authorized to “use[] the video conferencing functions of the video conference module 1302.” Ex.1005, [0171]. In this way, Abuan’s videoconference module (part of the “*function block*”) would determine whether the client application (“*determine that the superbloc application*”) may use the videoconference module’s services (“*is authorized to access functionality provided by the function block for the service*”). Ex.1003, ¶206.

[12.0] The method of claim 11, further comprising

See [11.0]. Ex.1003, ¶211.

[12.1] determining that the superblock application is authorized to access the service.

See [7.1]; Ex.1003, ¶212.

E. Ground 4

1. Guzman

Guzman describes a video-conferencing application with a semi-transparent overlay user interface superimposed on the live video display that centralizes account setup, authentication, contact management, and call initiation. Ex.1006, Abstract, [0002]–[0006], [0033], [0037], [0049]–[0051]. Through the overlay, a user can create an account or sign in to a service that validates credentials locally or via a remote server, after which the application retrieves and displays the user’s contacts for call placement. Ex.1006, [0044]–[0046], [0060]–[0063], [0074]–[0075], [0058], [0070]–[0071]. Guzman teaches an “auto-login” option that, once selected and validated, persists the user’s session by storing prior login information so subsequent launches bypass credential prompts and immediately load the user’s contacts and conferencing UI. Ex.1006 [0056]–[0058]. The system thus streamlines repeated access to a server-backed conferencing service by enabling persistent login behavior that avoids future authentication interruptions. Ex.1006, [0056]–[0059]; Ex.1003, ¶¶213-215.

Guzman is analogous art to the '046 patent. Guzman is in the same field of endeavor as the '046 patent, namely, the “manner in which functionality is accessed in certain environments, such as mobile device environments.” Ex.1001, 1:66-67; *see* Ex.1006, [0001]-[0002] (describing how “mobile phones[] allow users to engage in video conferences”). Guzman and the '046 patent also both generally relate to implementing functionality via software. *See* Ex.1001, Abstract (“providing additional functionality to existing software”); Ex.1006, [0221] (describing Guzman’s “Software Architecture” for “software running on a particular machine, such as... a handheld device”). Guzman and the '046 patent both describe at length features for facilitating videoconferencing. Ex.1001, Abstract, 4:43-51 (purpose of a function block is to provide functions including “audio, video,... conferencing, meetings, and/or other functions”); Ex.1006, Title (“Overlay for a video conferencing application”); Ex.1003, ¶216.

Guzman is also reasonably pertinent to a problem allegedly addressed by the '046 patent, namely, the need to provide functionality—such as communication functions—to a superblock application. Ex.1001, 3:58-4:15. Guzman provides a modular approach to software design for providing functionality to an application, illustrating in Fig. 22 “the software architecture of a video conferencing application 2200.” Ex.1006, [0221]. The application, which “is a stand-alone application,” includes various modules, including “user interface (UI) interaction

and generation module 2205, an account setup and verification module 2210, a contact management module 2215, an overlay generator 2220, a conference management module 2225, and a video display generator 2230.” Ex.1006, [0222]; Ex.1003, ¶217.

1. Reasons to combine Abuan and Guzman

A POSITA would have been motivated to combine the teachings of Abuan and Guzman for multiple reasons. Ex.1003, ¶¶226-231.

Abuan generally describes technology for facilitating videoconference communications. Ex.1005, Abstract. Abuan acknowledges that a videoconference between two parties requires the call recipient to accept the call. Ex.1005, [0117] (“turn the phone call into a video conference with the permission of the other party”). Abuan illustrates in Fig. 7 that initiating a video conference also requires the participation of a third intermediary device, termed a videoconference server. It would have been obvious to a POSITA for establishing a videoconference to also require the permission of the videoconference server, since the videoconference will also consume some of the limited resources available on the videoconference server. Ex.1003, ¶227. Allowing unrestricted and unlimited use of the videoconference feature would create the undesirable risk of the videoconference server becoming overwhelmed and failing, either intentionally (in a denial-of-service attack) or unintentionally. Ex.1003, ¶228; Material Fact #4; Ex.1037

(“denial of service attack (also, DoS attack) is an attack... that causes a loss of service to users, typically... by consuming the bandwidth of the victim network or overloading the computational resources of the victim system”); Ex.1038 (“denial of service (DOS) attacks, where... the server at the target IP address is overwhelmed by fake requests for service, and is either unable to service ‘real’ requests for service, or crashes due to overload”); Ex.1039, [0002] (“a denial of service attack, one or more attackers may send packets towards one or more servers, overwhelming the servers so that they are no longer able to serve their legitimate clients.”).

Guzman describes a “login procedure” in which a user desiring to access a service, such as a videoconference service, provides their username and password. Ex.1006, [0056]. To balance the need for verifying users’ login credentials with users’ desire to have quick and easy access to a server, Guzman further describes an “auto-login” option that lets the user “skip the login procedure” after they successfully complete it the first time. Ex.1006, [0056]; Ex.1003, ¶229.

A POSITA would have found it obvious to incorporate Guzman’s login techniques, including the auto-login option, into Abuan’s videoconferencing clients and server. Requiring a login was common at the time. Material Fact #1. Their combination is simply the integration of prior art elements (login procedures for accessing server-based services such as videoconferencing) according to known

methods (implemented using ordinary software programming techniques within the POSITA skill set) to yield predictable results (restricting Abuan's videoconference services to logged-in users). Ex.1003, ¶230. The combination of Abuan and Guzman is merely the application of a known technique (requiring user logins) to a known system (Abuan's videoconferencing architecture) ready for improvement (since Abuan does not discuss requiring login) to yield predictable results (restricting Abuan's videoconference services to logged-in users). Ex.1003, ¶¶229-230. Finally, the combination is also merely the use of known technique (requiring user logins) to improve similar devices (Abuan's videoconference clients and server) in the same way (limiting access to logged-in users). Ex.1003, ¶231.

Additional reasons to combine are provided in the detailed analysis below.

2. Dependent claims 8 and 13

[8.0] The non-transitory computer readable medium of claim 1 wherein the function block is further operative when executed by the processor to

See [1.0], [1.2]. Ex.1003, ¶¶218-219.

[8.1] provide authorization credentials to access the service, where the authorization credentials are received by the function block from the superblock application.

Abuan illustrates in Figure 7 a “video conference request messaging sequence 700” that initiates a videoconference. When a video conference client (e.g., client application 1365) sends a request to start a videoconference, a “Video

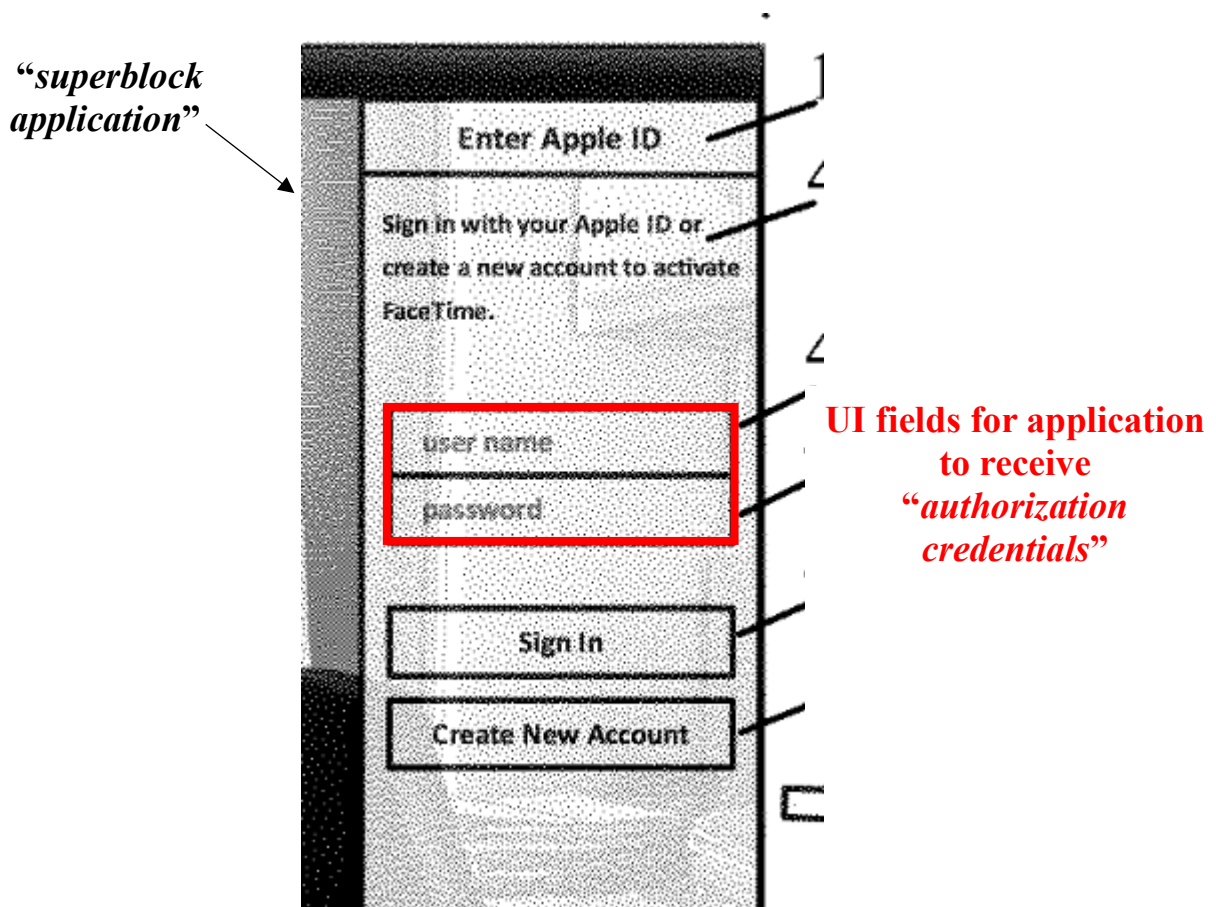
Conference Request” message is sent to a video conference server 715. Ex.1005, [0108]-[0109]; Fig. 7. Thus, Abuan’s Fig. 7 process assumes that a user is already logged into, and authorized to use, the video conference server.

It would have been obvious to a POSITA, however, for the video conference server to require users to login using a user name and password (*authorization credentials*) to use the server’s videoconferencing capabilities, because requiring a user to login was nearly universally employed by servers at the time. Ex.1003, ¶220; see Ex.1043, Fig. 6 (showing a “typical login screen” asking for “an account name 293 and a password 295”); Ex.1021, [0024] (“A user login typically includes a user name and password.”); Ex.1022, [0028] (describing “operations typically performed” during login for “authentication and authorization”); Ex.1023, [0002] (describing the “well known” login process in which “a server computer is responsible for granting access to a remote user based on a valid combination of userID and password”). Consistent with that understanding and with Abuan’s general architecture, Guzman describes a videoconferencing application for providing videoconference communications between two or more clients using a user name and password (*authorization credentials*). Ex.1006, Abstract, [0034].

Guzman describes a user providing “input information (e.g., username, password, contacts, etc.)” for an account in an “overlay display area” of a “video conferencing application.” Ex.1006, [0050]. Like Abuan’s video conference client

application (*the superblock application*), Guzman explains that its video conferencing application can be implemented using a “client” that interacts with a remote server. Ex.1006, [0221]; Ex.1003, ¶221.

Guzman describes a log-in procedure wherein the application receives, as a user input from UI (part of the “superblock application”) a username and password. Ex.1006, [0074], Fig. 3. With respect to Fig. 4 below, Guzman shows the overlay display area UI with two “entry fields” for “user name” and “password.”



Ex.1006, Fig. 4 (partial annotated)

“[W]hen the user selects the “Sign In” item 465, the video conferencing application verifies whether the login information entered by the user is valid.” Ex.1006, [0077]. Guzman explains that the user name and password entered through the application’s UI (“*received by the function block from the superblock application*”) can be verified by the video conferencing application itself, “or by accessing a remote server that stores login information.” Ex.1006, [0057]; Ex.1003, ¶¶221-223.

For instance, Guzman’s video conferencing application 2200 (analogous to Abuan’s client application 1365) includes “an account setup and verification module 2210” (part of “*the function block*”) that facilitates a user logging into the videoconferencing server. Ex.1006, [0222], [0230]. In one example, a “user inputs an account name and password which is **passed by the user interface interaction module 2205 to the account setup and verification module 2210.**” Ex.1006, [0230]. Thus, Guzman discloses a user interface interaction module 2205 (a “*function block*”) receiving, from the UI of the video conferencing application (“*superblock application*”), the user’s “authorization credentials,” and “*provid[ing]*” those credentials to the module 2210, or to a remote server for verification. When the credentials have been verified, the user’s contacts are loaded and made available so that the user can initiate a video conference. Ex.1006, [0058], Fig. 4. Thus, the user’s login credentials are “*provide[d]...to access*”

Guzman's videoconferencing service; Ex.1003, ¶¶224-225.

As explained above, a POSITA would have been motivated to combine Guzman's video conference login procedures with Abuan's video conference system and application because requiring a user to login was nearly universally employed by servers at the time. Ex.1003, ¶¶227-228. Abuan acknowledges the importance of restricting use of the video conference server's services to users who have permission and are authorized to use the service. Gatekeeping access to the video conference server and services helps to avoid accidentally overloading limited server resources, and to prevent malicious attacks (e.g., denial of service attacks). Ex.1003, ¶228.

Accordingly, Guzman discloses or renders obvious this limitation. Ex.1003, ¶¶220-231.

[13.0] The method of claim 11, further comprising

See [11.0]. Ex.1003, ¶232.

[13.1] determining that the service requires access to the resource, wherein the resource is external to the electronic device.

Abuan's Figure 7 shows that video conference initiation includes sending and receiving messages via the video conference server 715. Ex.1005, [0108]-[0109]; Fig. 7. The video conference server 715 (*the resource*) is shown to be a separate device from the "client device 705" running the video conference client application. Ex.1005, Fig. 7. Thus, video conference server 715 is "*external to the*

electronic device.”

As explained above for limitation [8.1], it would have been obvious to a POSITA for the video conference server to require users to login to use the server’s videoconferencing capabilities, as disclosed by Guzman. Ex.1003, ¶234; Ex.1006, Abstract, [0034], [0057]. As also explained for limitation [8.1], Guzman’s “account setup and verification module 2210” (part of “*the function block*”) facilitates a user logging into the videoconferencing server. Ex.1006, [0222], [0230]. Like Abuan’s videoconference server, Guzman’s remote videoconferencing server is an “*resource*” that is “*external to the electronic device.*” Ex.1003, ¶234.

Guzman’s login process allows the user to “enable auto-login,” and thus “skip the login procedure (e.g., the entering of a user name and password) the next time the user opens the video conferencing application.” Ex.1006, [0056]. Guzman also explains that the user’s login credentials (username and password) are verified either “through information stored on the device with the video conferencing application, or by accessing a remote server that stores login information for numerous users at different locations.” Ex.1006, [0057]. It would have been obvious to a POSITA that using a client application to start a videoconference would include determining whether the user’s credentials may be verified locally or if, instead, the user needs to log into the remote server (whether “*the service requires access to the resource*”). Ex.1003, ¶¶223-224, 229, 234-235.

Notably, the claim language does not specify how to determine whether “*access to a resource*” is required; thus, Guzman’s disclosure that user credentials are verified either locally or via contacting a remote server renders the claim limitation obvious because Guzman’s client application would obviously *determine[]* whether to verify the credentials locally or remotely. For example, where the user had previously logged in and selected the “auto-login” option, accessing the videoconferencing server to log in would not be necessary; otherwise, accessing the videoconferencing server to log in would be necessary. Ex.1003, ¶234.

As another example, it would have been obvious for the client application to confirm that the user has provided username and password values through the user interface. *See* Ex.1006, [0074]. Where the user failed to provide a password value, for example, it would be obvious for the client application to determine that contacting the remote server was unnecessary to confirm that the provided password (i.e., no password at all) was invalid. Ex.1003, ¶¶223-224, 234.

Guzman’s remote server storing login information may also provide other services, such as maintaining a user’s address book with the user’s contacts. Ex.1006, [0069] (“The address book may be stored locally or on a remote server (e.g., the same server... used to verify login information).”). Guzman expressly contemplates that the functionality of various modules—such as the “account setup

and verification module”—may be integrated into other modules or components.

Ex.1006, [0235]. It would have been obvious to integrate Guzman’s remote-server login functionality into Abuan’s videoconference server because, in the context of Abuan’s videoconference system shown in Fig. 7, the videoconference server is the only server. *See* Ex.1003, ¶¶220, 227. Thus, under the combined teachings of Abuan and Guzman, when a user’s client application determined that it was necessary to verify the user’s credentials with a remote server, the client application would send the credentials to videoconference server 715 for verification. Ex.1003, ¶¶225-231, 234-235.

XIII. Conclusion

Petitioner has established a reasonable likelihood that the Challenged Claims are unpatentable.

Respectfully submitted,

Dated: January 13, 2026
HAYNES AND BOONE, LLP
2801 N. Harwood St., Suite 2300
Dallas, Texas 75201
Customer No. 27683

/Theodore M. Foster/
Theodore M. Foster
Lead Counsel for Petitioner
Registration No. 57,456

XIV. Mandatory notices

A. Real party-in-interest

Pursuant to 37 C.F.R. § 42.8(b)(1), Petitioner certifies that the real party-in-interest is Cisco Systems, Inc.

B. Related matters

Pursuant to 37 C.F.R. § 42.8(b)(2), to the best knowledge of the Petitioner, the '046 Patent is or was involved in the following cases:

Case Heading	Number	Court	Filed
<i>Damaka, Inc. v Cisco Systems, Inc.</i>	2-25-cv-00593	EDTX	May 30, 2025
<i>Damaka, Inc. v. The Cigna Group</i>	2:25-cv-00594	EDTX	May 30, 2025

C. Lead and back-up counsel and service information

Lead Counsel

Theodore M. Foster
HAYNES AND BOONE, LLP
2801 Harwood St. Suite 2300
Dallas, TX 75201

Phone: (303) 382-6205
Fax: (214) 200-0853
ipr.theo.foster@haynesboone.com
USPTO Reg. No. 57,456

Back-up Counsel

Calmann J. Clements
HAYNES AND BOONE, LLP
2801 Harwood St. Suite 2300
Dallas, TX 75201

Phone: (972) 739-8638
Fax: (214) 200-0853
calmann.clements.ipr@haynesboone.com
USPTO Reg. No. 66,910

David L. McCombs
HAYNES AND BOONE, LLP
2801 Harwood St. Suite 2300
Dallas, TX 75201

Phone: (214) 651-5533
Fax: (214) 200-0853
david.mccombs.ipr@haynesboone.com
USPTO Reg. No. 32,271

Brett Bostrom
HAYNES AND BOONE, LLP
2801 Harwood St. Suite 2300
Dallas, TX 75201

Phone: (720) 484-3715
Fax: (214) 200-0853
brett.bostrom.ipr@haynesboone.com
USPTO Reg. No. 73,009

Dagim Tilahun
HAYNES AND BOONE, LLP
2801 Harwood St. Suite 2300
Dallas, TX 75201

Phone: (202) 654-4582
Fax: (214) 200-0853
dagim.tilahun.ipr@haynesboone.com
USPTO Reg. No. 75,722

Matthew Beck
HAYNES AND BOONE, LLP
2801 Harwood St. Suite 2300
Dallas, TX 75201

Phone: (202) 654-4573
Fax: (214) 200-0853
matthew.beck.ipr@haynesboone.com
USPTO Reg. No. 79,791

Please address all correspondence in this proceeding to lead and back-up counsel. Petitioner consents to service in this proceeding by email at the addresses above on condition that Patent Owner likewise consents to email service.

CLAIMS APPENDIX

- [1.0] A non-transitory computer readable medium embodying a computer program comprising instructions that include a function block and a superblock application,
- [1.1] wherein the function block is accessible only to the superblock application on an electronic device, and
- [1.2] wherein the function block is operative when executed by a processor of the electronic device to:
- [1.3] establish a connection with an external device via a network interface to provide a service requested by the superblock application without switching context from the superblock application to another application on the electronic device.
- [2.0] The non-transitory computer readable medium of claim 1 wherein the function block is further operative when executed by the processor to:
- [2.1] receive from the superblock application a request for the service,
- [2.2] wherein the service is a communication session, and
- [2.3] wherein the function block is configured to provide functionality for the communication session to the superblock application; and
- [2.4] notify the superblock application that the communication session is being requested.

- [3.0] The non-transitory computer readable medium of claim 2
- [3.1] wherein the steps of receiving, notifying, and establishing are performed without switching context from the superblock application to another application on the electronic device.
- [4.0] The non-transitory computer readable medium of claim 2 wherein the function block is further operative when executed by the processor to
- [4.1] determine whether the function block has a capability to perform any rendering needed by the communication session.
- [5.0] The non-transitory computer readable medium of claim 1 wherein the function block is further operative when executed by the processor to
- [5.1] enable a user of the electronic device to collaborate on at least one of a document, a file, or an application with a user on another electronic device.
- [6.0] The non-transitory computer readable medium of claim 1 wherein the function block is further operative when executed by the processor to
- [6.1] provide at least one configuration parameter for the function block,
- [6.2] wherein the configuration parameter enables the function block to be configured after the function block is distributed as part of the superblock application.
- [7.0] The non-transitory computer readable medium of claim 1 wherein the

- function block is further operative when executed by the processor to
- [7.1] determine that the superblock application is authorized to access functionality provided by the function block for the service.
- [8.0] The non-transitory computer readable medium of claim 1 wherein the function block is further operative when executed by the processor to
- [8.1] provide authorization credentials to access the service, where the authorization credentials are received by the function block from the superblock application.
- [9.0] The non-transitory computer readable medium of claim 1 wherein the function block is further operative when executed by the processor to
- [9.1] receive a request for the service from the superblock application as an application programming interface (API) call from the superblock application to the function block.
- [10.0] The non-transitory computer readable medium of claim 1 wherein the function block is further operative when executed by the processor to
- [10.1] determine whether the service is available on the external device.
- [11.0] A method implemented in a function block associated with an electronic device having a network interface, comprising:
- [11.1] accessing a resource via the network interface to provide a service to a superblock application on the electronic device,

- [11.2] wherein the service is not otherwise supported by the superblock application, and
- [11.3] wherein the function block is accessible only to the superblock application on the electronic device; and
- [11.4] providing the service to the superblock application without switching context from the superblock application to another application on the electronic device.
- [12.0] The method of claim 11, further comprising
- [12.1] determining that the superblock application is authorized to access the service.
- [13.0] The method of claim 11, further comprising
- [13.1] determining that the service requires access to the resource, wherein the resource is external to the electronic device.
- [14.0] The method of claim 11, further comprising
- [14.1] operating as a bridge between the superblock application and the resource.
- [15.0] The method of claim 11, further comprising
- [15.1] initializing the function block when the superblock application is loaded and executed.
- [16.0] The method of claim 11, further comprising
- [16.1] providing at least one configuration parameter for the function block,

- [16.2] wherein the configuration parameter enables the function block to be configured after the function block is distributed as part of the superblock application.
- [17.0] The method of claim 11, further comprising
 - [17.1] determining whether the function block is capable of performing any rendering needed by the service.
- [18.0] The method of claim 11, further comprising
 - [18.1] enabling a user of the electronic device to collaborate on at least one of a document, a file, or an application with a user on another electronic device.
- [19.0] The method of claim 11, further comprising
 - [19.1] receiving a request for the service from the superblock application, wherein the request is received as an application programming interface (API) call from the superblock application to the function block.
- [20.0] The method of claim 11, further comprising
 - [20.1] determining whether the resource is available.

CERTIFICATE OF WORD COUNT

Pursuant to 37 C.F.R. § 42.24(d), Petitioner hereby certifies, in accordance with and reliance on the word count provided by the word-processing system used to prepare this Petition, that the number of words in this paper is 13,325. Pursuant to 37 C.F.R. § 42.24(d), this word count excludes the table of contents, table of authorities, mandatory notices under § 42.8, certificate of service, certificate of word count, appendix of exhibits, and any claim listing.

Dated: January 13, 2026

/Theodore M. Foster/

Theodore M. Foster

Lead Counsel for Petitioner

Registration No. 57,456

CERTIFICATE OF SERVICE

Pursuant to 37 C.F.R. § 42.6(e), the undersigned certifies that a true and correct copy of the foregoing was served on the Patent Owner as detailed below:

<i>Date of service</i>	January 13, 2026
<i>Manner of service</i>	FedEx
<i>Documents served</i>	PETITION FOR <i>INTER PARTES</i> REVIEW UNDER 35 U.S.C. § 312 AND 37 C.F.R. § 42.104; PETITIONER’S POWER OF ATTORNEY; PETITIONER’S EXHIBIT LIST AND EXHIBITS Ex.1001-1023, Ex.1029-1052
<i>Persons Served</i>	Munck Wilson Mandala L.L.P 2000 McKinney Ave Ste 1900 Dallas, TX 75201

/Theodore M. Foster/

Theodore M. Foster
Lead Counsel for Petitioner
Registration No. 57,456