

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

CISCO SYSTEMS, INC.,
Petitioner

IPR2023-00809
U.S. Patent No. 9,883,048

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

TABLE OF CONTENTS

I.	Introduction.....	6
II.	Grounds for standing	7
III.	Note.....	7
IV.	Technology background	7
V.	Summary of the '048 patent	7
VI.	Prosecution history	9
VII.	Effective priority date of the '048 patent	10
VIII.	Level of ordinary skill in the art	11
IX.	Claim construction.....	11
X.	Relief requested and the reasons for the requested relief.....	13
XI.	Identification of how the claims are unpatentable.....	13
	A. Challenged claims	13
	B. Statutory grounds for challenges.....	13
	C. Ground 1: Claims 13, 14, and 17 are obvious under 35 U.S.C. § 103 over de Castro and Robb	15
	1. De Castro.....	15
	2. Robb	18
	3. Reasons to combine de Castro and Robb.....	21
	4. Claim 13	26
	5. Claim 14.....	67

6.	Claim 17	72
XII.	Discretionary denial is not appropriate.....	74
A.	No basis for 35 U.S.C. § 325(d) denial	74
B.	Discretionary denial under the <i>Fintiv</i> factors is not appropriate.....	75
1.	Whether evidence exists that a stay may be granted if a proceeding is instituted	75
2.	Proximity of the court’s trial date to the board’s projected statutory deadline for a final written decision	75
3.	Investment in the parallel proceeding.....	76
4.	Overlap between issues raised in the petition and in the parallel proceeding.....	77
5.	Whether the petitioner is a defendant	77
6.	Other circumstances that impact the Board’s exercise of discretion, including the merits.....	78
C.	Discretionary denial under <i>General Plastic</i> is not appropriate.....	78
XIII.	Conclusion	79
XIV.	Mandatory notices	80
A.	Real party-in-interest.....	80
B.	Related matters	80
C.	Lead and back-up counsel and service information.....	80
CERTIFICATE OF WORD COUNT		82
CERTIFICATE OF SERVICE		83

Petitioner’s Exhibit List

Ex.1001	U.S. Pat. No. 9,883,048 to Paul et al. (“’048 patent”)
Ex.1002	Prosecution History of U.S. Pat. No. 9,883,048
Ex.1003	Declaration of Henry Houh under 37 C.F.R. § 1.68
Ex.1004	<i>Curriculum Vitae</i> of Henry Houh
Ex.1005	U.S. Pat. Pub. No. 2012/0016932 to de Castro et al.
Ex.1006	U.S. Pat. No. 8,126,722 to Robb et al.
Ex.1007	Zapier, “API Reference – Developers,” Jan. 23, 2013 web archive of https://zapier.com/developer/reference/ , available at Wayback Machine on Oct. 7, 2022
Ex.1008	Zapier, “Integrate your Web Services,” Jan. 23, 2013 web archive of https://zapier.com/developer/reference/ , available at Wayback Machine on Oct. 7, 2022
Ex.1009	Ben Langfeld, Jose de Castro, “XEP-0327: Rayo,” XMPP Standards Foundation (<i>archived</i> August 18, 2013)
Ex.1010	U.S. Pat. No. 7,593,980 to Marascio et al.
Ex.1011	U.S. Pat. No. 7,496,054 to Taylor
Ex.1012	U.S. Pat. No. 9,557,889 to Raleigh et al.
Ex.1013	Declaration of Jose de Castro under 37 C.F.R. § 1.68
Ex.1014	Affidavit of Authenticity for Ex.1007 and Ex.1008
Ex.1015	Exhibit A to Ex.1014
Ex.1016	Affidavit of Authenticity for Ex.1009
Ex.1017	U.S. District Courts - National Judicial Caseload Profile (December 2022), available at: https://www.uscourts.gov/statistics/table/na/federal-court-management-statistics/2022/06/30-2

Ex.1018	Docket Control Order, <i>Zilkr Cloud Technologies, LLC v. Cisco Systems, Inc.</i> , No. 2-22-cv-00166-JRG-RSP, ECF No. 31 (E.D. Tex., Oct. 13, 2022)
Ex.1019	Notice of Disclosures, <i>Zilkr Cloud Technologies, LLC v. Cisco Systems, Inc.</i> , No. 2-22-cv-00166-JRG-RSP (E.D. Tex., Sep. 22, 2022)
Ex.1020	Original Complaint for Patent Infringement, <i>Zilkr Cloud Technologies, LLC v. Cisco Systems, Inc.</i> , No. 2-22-cv-00166-JRG-RSP, ECF No. 1 (E.D. Tex., May 20, 2022)
Ex.1021	Director Katherine K. Vidal, “Interim Procedure for Discretionary Denials in AIA Post-Grant Proceedings with Parallel District Court Litigation,” June 21, 2022
Ex.1022	JAMES McCaffrey, SOFTWARE TESTING: FUNDAMENTAL PRINCIPLES AND ESSENTIAL KNOWLEDGE (2009)
Ex.1023	CEM KANER, JAMES BACH, & BRET PETTICHORD, LESSONS LEARNED IN SOFTWARE TESTING: A CONTEXT-DRIVEN APPROACH (2002)
Ex.1024	Microsoft Press Computer Dictionary (3d ed. 1997)
Ex.1025	Newton’s Telecom Dictionary (18th ed. 2002)
Ex.1026	Copyright Record for Ex.1024
Ex.1027	Copyright Record for Ex.1025
Ex.1028	PACER Calendar Events Report for Judge Rodney Gilstrap, January 22, 2024.
Ex.1029	SRINIVASAN DESIKAN & GOPALASWAMY RAMESH, SOFTWARE TESTING: PRINCIPLES AND PRACTICES (2011)
Ex.1030	U.S. Pat. No. 8,600,836 to Collins.
Ex.1031	U.S. Pat. Pub. No. 2007/0033194 to Srinivas <i>et al.</i>

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 13, 14, and 17 (hereinafter, the “Challenged Claims”) of U.S. Patent No. 9,883,048 (the “’048 patent,” Ex.1001).

The ’048 patent is directed to providing multiple third-party telecommunications applications and services to add to existing telephone service. Ex.1001, 1:31-33, 8:50-54. Such additional applications and services include controlling phone calls over the internet, messaging, and web services. Ex.1001, 9:38-49. The ’048 patent describes integrating third-party application interfaces and a telephone service provider interface into a common interface and publishing information about the applications in a catalog for users. Ex.1001, 12:9-14, 58:22-55. However, as shown below and confirmed in the Declaration of Dr. Henry Houh (Ex.1003), these concepts were well known in the prior art before the ’048 patent’s earliest priority date. E.g., Ex.1005, Fig. 14, [0038]-[0041], [0138]-[0142]; Ex.1006, 34:15-27.

The prior art presented in this Petition renders obvious the Challenged Claims.

II. GROUNDS FOR STANDING

Petitioner certifies that the '048 patent is eligible for IPR and that Petitioner is not barred or estopped from requesting IPR challenging the patent claims. 37 C.F.R. § 42.104(a).

III. NOTE

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

IV. TECHNOLOGY BACKGROUND

Dr. Houh's Declaration (Ex.1003) provides a technology background. Ex.1003, ¶¶39-47.

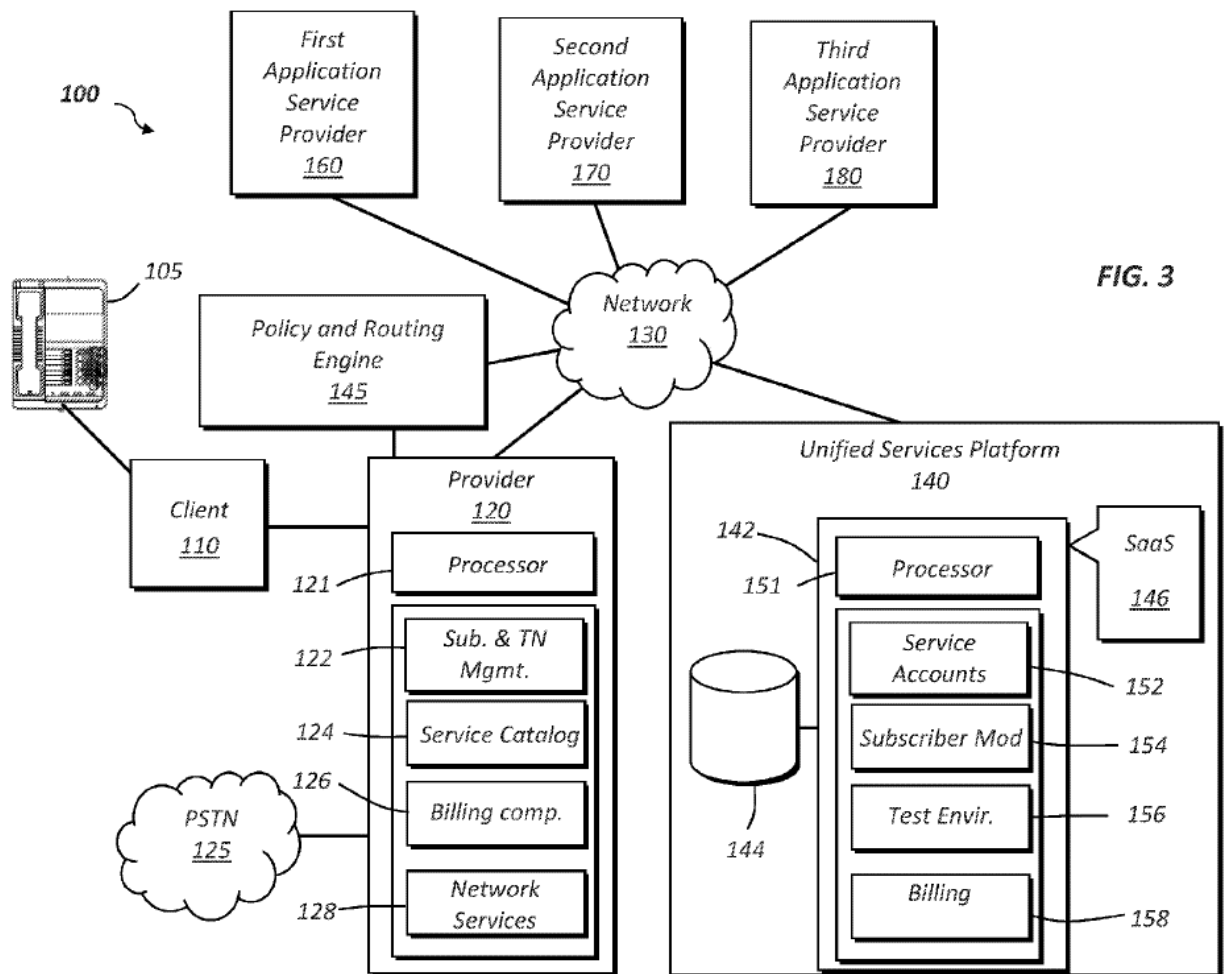
V. SUMMARY OF THE '048 PATENT

According to the '048 patent, “the telephone number remains a valuable identifier for providing telephone services.” Ex.1001, 1:28-30. Telephone service customers sometimes “wish to add services and applications to their existing telephone service.” Ex.1001, 1:31-33. Examples of third-party telecommunications services and applications include session initiation protocol (SIP) for controlling phone call sessions over internet networks, and applications for messaging, collaboration, and web services. Ex.1001, 9:38-49. “However, many services and applications are supplied by third-party providers in different ecosystems,” and as a result, they use different application programming interfaces (APIs) that may be

incompatible with each other. Ex.1001, 1:31-35, 58:45-48. The '048 patent alleges that “new techniques for integrating advanced services and applications with legacy telephone service are desired.” Ex.1001, 1:39-41; Ex.1003, ¶¶48-49.

The proposed solution, according to the '048 patent, is “integrating [a] plurality of API interfaces” to “provide services for use in association with a telephone number of a subscriber.” Ex.1001, 1:45-59. For instance, “two of the API interfaces are different, and at least one of the plurality of API interfaces from the plurality of application service provider systems is not compatible with the [network] provider API interface.” Ex.1001, Abstract, 1:59-63. To address such incompatibilities, the '048 patent merely describes “integrating [] each of the plurality of API interfaces with a unified services platform API interface” (Ex.1001, 45:48-67, 56:19-37), which “provide[s] a common interface between the application service providers” (Ex.1001, 13:35-42). Ex.1003, ¶¶50-51.

Figure 3 (below) illustrates unified services platform 140—which provides a common or unified API for the different APIs—connected via network 130 to client or device 110, telephone service provider 120, and application service providers 160, 170, 180. Ex.1001, Fig.3, 10:24-32, 10:51-53, 13:35-38.



Ex.1001, Fig.3

These and the other concepts recited in the Challenged Claims were known in the prior art before to the earliest priority date of the '048 patent. E.g., Ex.1005, [0038], [0119]; Ex.1006, 33:27-29; Ex.1003, ¶¶52-53.

VI. PROSECUTION HISTORY

The '048 patent's earliest priority claim is to provisional application 61/888,856, filed on October 9, 2013. The '048 patent was filed on April 8, 2016 as Appl. No. 15/094,924. The first Office action rejected independent claim 15

(later renumbered as independent claim 13 in the issued '048 patent), among other claims, as obvious over several references. The first Office action found allowable subject matter in dependent claim 16, which recites:

...publishing, after testing , [sic] the definition of the service, the definition of the plan, and the one or more definitions of actions for each application service provider in a provisioning catalog.

Ex.1002, 66, 96.

Applicant amended independent claim 15 to incorporate this limitation and cancelled dependent claim 16. Ex.1002, 66, 70. The Patent Office then issued a notice of allowance on October 11, 2017 (and a corrected notice of allowance on November 7, 2017) and the '048 patent issued on January 30, 2018 with independent claim 15 renumbered as independent claim 13. Ex.1002, 9.

VII. EFFECTIVE PRIORITY DATE OF THE '048 PATENT

The '048 patent claims priority to a provisional application filed October 9, 2013, and all of the cited and relied-upon prior art predates this date. 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 '048 patent in this or another forum where the issue is relevant.

VIII. LEVEL OF ORDINARY SKILL IN THE ART

A Person of Ordinary Skill in The Art (“POSITA”) in October 2013 would have had a working knowledge of the telecommunications and networking art that is pertinent to the ’048 patent. A POSITA would have had a bachelor’s degree in electrical engineering, computer science, computer engineering, or an equivalent, and two years of relevant experience in telecommunications and networking. Lack of professional experience can be remedied by additional education, and vice versa. Ex.1003, ¶¶32-34.

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). Petitioner submits that, for the purposes of this proceeding and the grounds presented herein, no claim term requires express construction. *Nidec Motor Corp. v. Zhongshan Broad Ocean Motor Co.*, 868 F.3d 1013, 1017 (Fed. Cir. 2017).

While an express construction is not necessary for the purposes of this proceeding, the ’048 specification provides an example of the claim term “*definition of a service*” and that is relevant to the prior art discussed below: “The services can be defined through the use of catalog data such as a **service name**,... **description**,... **availability**, and the like.” Ex.1001, 15:65-16:1. Therefore, the

'048 patent describes a service name, description, and availability as examples of “*a definition of a service.*” Ex.1001, 15:65-16:1; Ex.1003, ¶¶54-55.

Also, while an express construction is not necessary for the purposes of this proceeding, the '048 patent specification also provides an example of the claim term “*definition of [a] plan*” that is relevant to the prior art below: “The plans comprise information regarding pricing, features, and/or **service limits associated with the service** offered by the application service provider 160.” Ex.1001, 16:5-8. Therefore, the '048 patent describes service limits associated with the service is an example of a “*definition of [a] plan.*” Ex.1001, 16:5-8; Ex.1003, ¶¶54-55.¹

Finally, petitioner notes that the Applicant made arguments regarding the scope of test environment in prosecution. Citing the specification of the '048 patent, the Applicant explained that “the test environment ... may respond in a manner that is the same or similar to the live implementation ... without allow [sic] the interactions to be released.” Ex.1002, 72 (quoting '048 patent, 14:65-15:1). The Applicant then argued that “the test environment is the same as the unified services platform without the actual external connections or communications.” Ex.1002, 72.

¹ Claim 13 actually recites “*the definition of the plan.*” Cisco may challenge this language for indefiniteness in another forum that allows for such challenges.

To the extent the Board construes the term in this way, the prior art renders the claim obvious applying this construction. Ex.1003, ¶134.

X. RELIEF REQUESTED AND THE REASONS FOR THE REQUESTED RELIEF

Petitioner asks that the Board institute an IPR trial and cancel the Challenged Claims.

XI. IDENTIFICATION OF HOW THE CLAIMS ARE UNPATENTABLE

A. Challenged claims

Petitioner challenges claims 13, 14, and 17. The Challenged Claims are asserted in *Zilkr Cloud Technologies, LLC v. Cisco Systems, Inc.*, No. 2-22-cv-00166 (E.D. Tex. filed May 20, 2022). A finding that the Challenged Claims are unpatentable in this proceeding will resolve the parties’ dispute and obviate the need for a trial regarding the ’048 patent in the co-pending litigation, substantially reducing the time and expense of that litigation for all parties.

B. Statutory grounds for challenges

Ground	Claims	Basis
#1	13, 14, and 17	35 U.S.C. § 103 over de Castro in view of Robb

U.S. Pat. Pub. No. 2012/0016932 to de Castro et al. (Ex.1005, “de Castro”) published on January 19, 2012.

U.S. Pat. No. 8,126,722 to Robb (Ex.1006, “Robb”) issued on February 28, 2012.

De Castro and Robb are prior art under 35 U.S.C. § 102(a)(1).²

Petitioner’s obviousness Ground relies on the combined teachings of the references as understood by a POSITA and not on a physical incorporation of elements from one reference into another. *See In re Mouttet*, 686 F.3d 1322, 1332 (Fed. Cir. 2012); Ex.1003, ¶¶71-83.

Petitioner cites 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. Exhibit 1009 is an XMPP Specification document referred by de Castro as the Rayo Specification. Its prior art status is confirmed by Ex.1013 and Ex.1016. Exhibits 1007 and 1008 are web pages from Zapier.com. Their prior art status is confirmed by Ex.1014 and Ex.1015. Ex.1024 and Ex.1025 are technical dictionaries for telecom and computer terms. The prior art status of Ex.1024 is confirmed by Ex.1026. The prior art status of Ex.1025 is confirmed by Ex.1027.

² Petitioner notes that the “Swear Behind” provisions of 37 C.F.R. § 1.131 do not apply to the ’048 patent. *See* MPEP §§ 715, 2159.02.

The Petition’s use of supporting tertiary evidence has been repeatedly approved. *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).

C. Ground 1: Claims 13, 14, and 17 are obvious under 35 U.S.C. § 103 over de Castro and Robb

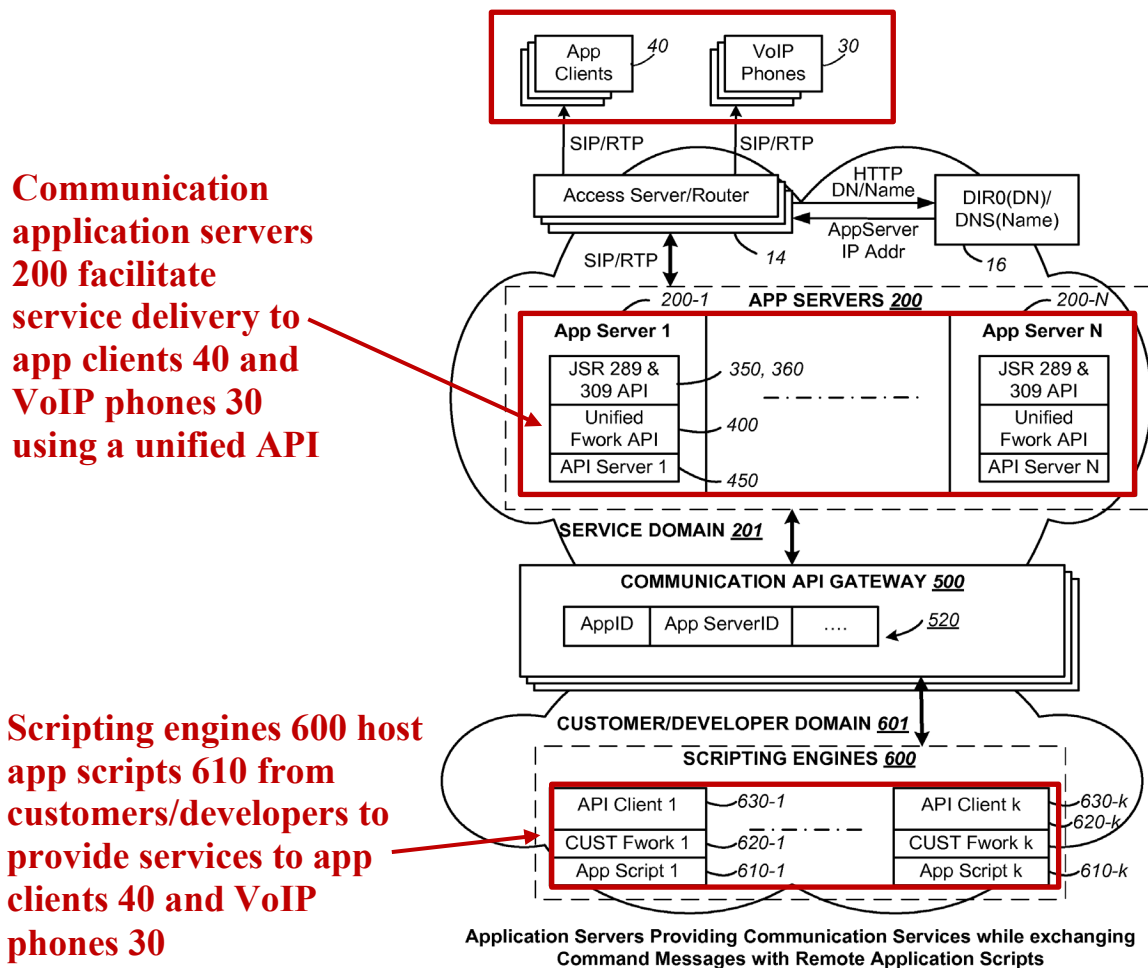
1. De Castro

De Castro describes “a plurality of communication application servers 200” that “allow[] application clients 40 ... to access the application hosted in the communication application servers.” Ex.1005, [0115]. A “client” includes a user device utilizing a telephony service, such as a voice-over-IP (VoIP) phone. Ex.1005, [0145], [0116]; Ex.1003, ¶59.

De Castro’s communication application servers communicate with “a plurality of scripting engines 600” that are used to “host and process the customers’ application scripts 610.” Ex.1005, [0119]. Note that de Castro carefully distinguishes the terms “customer” and “client.” Whereas “client” refers to an end user who uses a service (or to an end user’s device), the term “customer” refers to a

“developer” that provides a service by coding an application script. Ex.1005, [0071], [0113]-[0114]; Ex.1003, ¶60.

Fig.14 below illustrates VoIP phones 30 and application clients 40 connected to scripting engines 600 via communication application servers 200; Ex.1003, ¶61.



Ex.1005, Fig.14 (annotated); Ex.1003, ¶61

The communication application servers 200 use “a messaging API protocol [that] enables an application script to pass commands remotely to the communication application server to operate it.” Ex.1005, [0038]. This messaging API protocol or “unified API” allows a “customer using the communication services [to] code the application script in a preferred programming language using a custom framework providing a set of customer-specific libraries.” Ex.1005, [0038], [0041]. An example of the messaging API protocol is the “Rayo protocol” which “primarily deals with calls” and telecommunications services associated with a user’s telephone number. Ex.1005, [0138]-[0141]. Ex.1003, ¶¶62-63.

De Castro is analogous art to the ’048 patent and is from the same field of endeavor: provisioning many different applications from different developers to end users through a unified provider system. *See* Ex.1005, [0038]-[0039], [0066]; Ex.1001, Abstract. For example, de Castro describes a “unified framework” that is used to provide application clients access to a variety of communication applications. Ex.1005, Abstract. Similarly, the ’048 patent provides a unified services platform to access a variety of services and applications from multiple different ecosystems. *See* Ex.1001, Abstract. Both de Castro and the ’048 patent describe using a common API protocol for the various service providers to execute their communication applications. *See* Ex.1001, 4:57-62; Ex.1005, Abstract. De Castro is also reasonably pertinent to the particular problem of the ’048 patent.

Specifically, the '048 patent explains that “new techniques for integrating advanced services and applications with legacy telephone service are desired.” Ex.1001, 1:39-41. De Castro addresses this exact problem with its unified API and API server. *See* Ex.1005, Abstract. For example, de Castro describes how service providers may use their own APIs to provide communications services to clients by telephone. *See* Ex.1001, Abstract, 11:33-42; Ex.1005, [0007], [0022]; Ex.1003, ¶64.

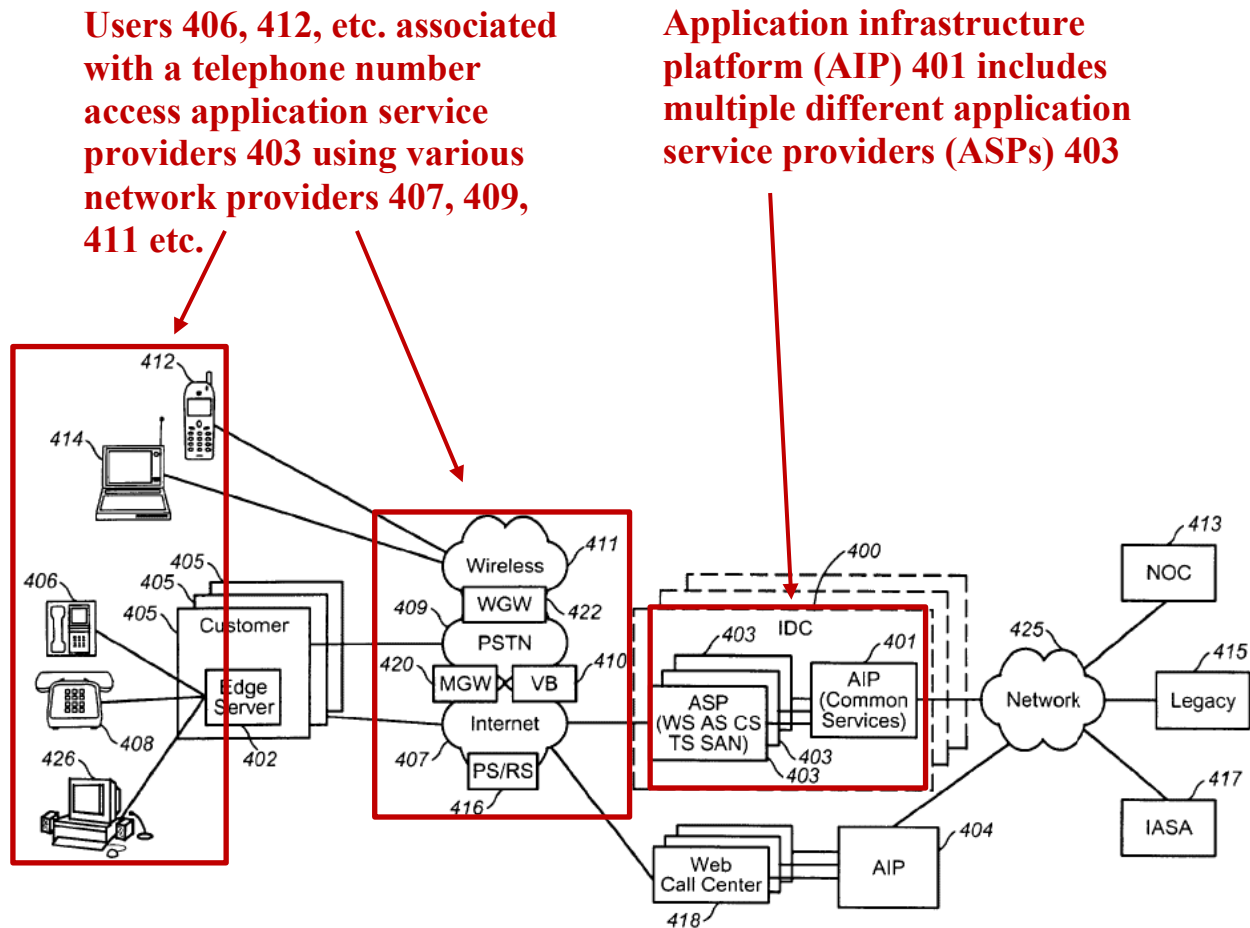
2. Robb

Robb relates to delivering “services and data to end user devices” such as telephones and personal digital assistants. Ex.1006, Fig. 5, Abstract, 9:43-65, 12:16-24, 14:40-48, 16:25-36. Robb describes “an application infrastructure platform (AIP) ... including multiple [application service provider] ASP systems providing application services to customers” (e.g., users³) who are associated with telephone numbers. Ex.1006, Abstract, Fig.5; Ex.1003, ¶65.

Fig. 5 depicts telephone users (e.g., 406 and 412) accessing application infrastructure platform 401 that includes multiple different application service

³ Note that while de Castro uses the term “customer” to refer to an application developer, Robb uses the term “customer” to refer to an end user who accesses telephony services. *See* Ex.1005, [0039], [0041], [0113]-[0114]; Ex.1006, 2:48-49.

providers 403 via different network providers (e.g., Internet 407, and public switched telephone network 409). Ex.1006, Fig. 5, 15:56-16:9; Ex.1003, ¶66.



Ex.1006, Fig.5 (annotated); Ex.1003, ¶66

Robb teaches that “the AIP 114 provides a collection of applications and services to the hosted ASPs 102.” Ex.1006, 7:28-32. To make this collection of applications available to other application service providers (ASPs), Robb explains that an application infrastructure platform (AIP) may “aggregate its own applications and services with those of both external ASPs/Internet Service

Vendors (ISVs) and its own ASPs into a ‘Product Catalog’.” Ex.1006, 7:53-55.

Application service providers can then select from the cataloged applications and services to “repackage and resell to their customers as corporate portals.” Ex.1006, 7:28-32. Robb explains that this service aggregation and customization technique “enables service providers to leverage more customers by wholesaling or leasing to other service providers than they can address on their own.” Ex.1006, 26:47-49.

Robb refers to the selected and customized services in the customer specific portals as “service packages,” where a service package includes “an aggregation of services for a user role” and “SLA [service-level agreement] terms and conditions to the service.” Ex.1006, 31:40-41, 34:16-17, 34:25-27; Ex.1003, ¶¶67-68.

Robb also describes a “Staging and Test Bed environment 503.” Ex.1006, 17:5-9. The staging and test bed environment is used to “integrate new applications onto systems, stage them for scaled system network test, and finally prepare them for automatic software deployment to AIP sites.” Ex.1006, 17:5-9. The testing process described in Robb includes testing performance of a rules engine that “evaluates events using a rule base and, when the criteria for a rule is met, ... initiat[ing] ... a specific work flow.” Ex.1006, 10:52-55. After testing for each new application, access to the new application is provided to ASPs through the AIP. *See* Ex.1006, 7:2930, 38-41, 17:5-9; Ex.1003, ¶¶69.

Robb is analogous art to the '048 patent and is from the same field of endeavor: provisioning many different applications from different developers to end users through a unified provider system. Ex.1006, Abstract, 6:24-25; Ex.1001, Abstract. For example, Robb describes an “application infrastructure platform” that is used to provide application clients access to “multiple application services.” Ex.1006, Abstract. Similarly, the '048 patent provides a unified services enabler platform to access a variety of services and applications from multiple different ecosystems. *See* Ex.1001, Abstract. Both Robb and the '048 patent describe using a common API protocol or platform for the various service providers to execute their communication applications. *See* Ex.1001, 4:57-62; Ex.1006, 15:56-16:9. Robb is also reasonably pertinent to the particular problem of the '048 patent. For example, Robb describes how application service providers may integrate their services into a platform to provide communications services to clients by telephone. *See* Ex.1001, Abstract, 11:33-42; Ex.1006, 6:24-35. Robb also describes how service providers may select from cataloged services and publish a selection of services on a portal for telephone clients. *See* Ex.1001, 12:9-14; Ex.1006, 34:16-17; Ex.1003, ¶70.

3. Reasons to combine de Castro and Robb

A POSITA considering the teachings of de Castro in October 2013 would have also considered the teachings of Robb, as they are analogous art and pertain

to the same field of endeavor. *See, e.g.*, Ex.1005, [0039]; Ex.1006, Fig.5. Each of these references describes integrating multiple telecommunications application services and network providers for end users associated with a telephone number. *See, e.g.*, Ex.1005, Fig.14, [0139]-[0141]; Ex.1006, Fig.5, 9:43-50; Ex.1003, ¶71.

It would have been obvious, beneficial, and predictable to a POSITA at the time to utilize Robb's teachings regarding aggregating telecommunication services in a product catalog and repackaging and reselling the cataloged services for use on corporate portals in combination with de Castro's system. Ex.1003, ¶72. For example, a POSITA would have appreciated the benefits of deploying various tailored combinations of de Castro's Rayo-based services using Robb's product catalog and corporate portal techniques. Ex.1003, ¶¶74-75, 78. Indeed, de Castro already contemplates: (1) developers coding application scripts; (2) these scripts including commands that are expressed in standardized Rayo protocol messages; and (3) simplifying the telecommunications application development process to allow the developer to "focus on the business logic." Ex.1005, [0038], [0041], [0086], [0098]; Ex.1003, ¶73. For instance, in de Castro's system, developers can "code the application script in a preferred programming language," which is then "expressed in terms of the finite set of commands" of a unified API, e.g., the Rayo protocol. Ex.1005, [0041], [0138]-[0141]; Ex.1003, ¶73. De Castro also teaches that the unified API (Rayo protocol) "allows application clients 40 such as, for

example, VoIP phones to access the applications hosted in the communication application servers.” Ex.1005, [0115]; Ex.1003, ¶73.

Robb provides details on how a platform that aggregates multiple telecommunications services from different providers (like de Castro’s communication application server) publishes the aggregated services to a product catalog for selection and customization by other service providers. Ex.1003, ¶76. Robb also explains how service providers use a “corporate portal” to leverage the aggregation of services in the product catalog to customize, repackage, and resell service combinations to their end users. Ex.1006, 26:47-49, 34:16-17, 34:24-28, 35:9-10; Ex.1003, ¶76. For instance, Robb describes publishing “service packs” (i.e., application service software code and licensing terms) on a corporate portal. Ex.1006, 34:16-17, 34:24-28; Ex.1003, ¶76. As described in more detail in sections XI.C.4.[13.8], the “product catalog” is the equivalent of the claimed “*provisioning catalog*” of [13.8]. A POSITA would have found the combination obvious, where Rayo-protocol services are first made available to other developers via a product catalog, and then customized and repackaged for de Castro’s clients via a portal website educating users about the available services, per Robb. Ex.1003, ¶76.

Robb also provides details on how testing may be performed to ensure the accuracy of application software prior to deployment. Ex.1006, 7:38-41, 16:59-17:30, 33:49-51, 34:29-31. For example, Robb provides “a tested and integrated

environment” that a POSITA at the time of the invention would have recognized as superior to an untested environment. Ex.1006, 8:41-44; Ex.1003, ¶77.

A POSITA would also have recognized the benefits of publishing the different application services provided by (or available using) de Castro’s system in a product catalog, per Robb. *See* Ex.1005, Fig.14, [0041], [0138]-[0141]; Ex.1006, Fig.5, 34:16-17, 34:24-28; Ex.1003, ¶79. For instance, a POSITA would have recognized that the combination beneficially allows different developers or service providers to “leverage more customers by wholesaling or leasing to other service providers than they can address on their own.’ Ex.1006, 26:47-49; Ex.1003, ¶78. Further, a POSITA would have recognized the advantages of using Robb’s product catalog to enable service providers to “provide customized content to the end user” by repackaging selected services from the product catalog through a customer-specific portal.” Ex.1006, 9:66-10:1. This advantageously expands the range of telecommunication services and service packages available to both service providers and end users. Ex.1006, 6:28-32. The combination thus beneficially allows service providers “to augment their own offerings presented as blended services through a unified corporate portal.” Ex.1006, 6:32-37; Ex.1003, ¶¶78-79.

A POSITA also would have recognized that using a corporate portal that publishes service packs, per Robb, would be beneficial for de Castro’s system because the “services offered through a corporate portal have also grown more

robust.” Ex.1006, 6:53-57. For instance, Robb explains that “[i]nstead of purchasing licenses (seats) for employees to use various software packages that are installed on their personal computers, many corporations find it less burdensome to lease software on a per use basis.” Ex.1006, 6:53-57. Robb’s technique thus beneficially provides de Castro’s system a flexible and user-friendly implementation—because as Robb explains, “portals can have a specific focus, such as personal portals, business-to-business (B2B) portals, business-to-employee (B2E) portals, and wireless portals. Portals filter, aggregate, and present the data to the end user.” Ex.1006, 10:1-5; Ex.1003, ¶79. In the context of telephony services offered directly to an end user (e.g., outside the corporate context), it would have been obvious for a service provider to publish available services to a personal portal where the user could access those additional services. Ex.1003, ¶79.

The combination of de Castro and Robb is merely the ordinary use of common techniques (aggregating services in a product catalog and publishing services on a portal, per Robb) in the similar unified communication application system of de Castro to yield predictable results (Rayo-protocol application services available in a product catalog and a portal). Ex.1003, ¶80. Robb’s product catalog and portal techniques were also specifically intended to be implemented in servers integrating multiple telecommunications applications (such as de Castro’s). *See* Ex.1005, Fig.14, [0038], [0041]; Ex.1006, Fig.5, Fig. 16, 27:50-58, 28:4-17, 34:16-

17, 34:24-28; Ex.1003, ¶80. All of these reasons, individually and collectively, would have motivated a POSITA to combine the references to arrive at the invention recited in the Challenged Claims. Ex.1003, ¶80.

The results would have been predictable and there would have been a reasonable expectation of success in the combination given that both de Castro and Robb describe similar concepts (e.g., telephone users accessing different application services over telecommunications and internet networks). *See* Ex.1005, Fig.14; Ex.1006, Fig.5; Ex.1003, ¶81. Furthermore, aggregating telecommunications services in a product catalog and publishing services in a corporate portal was long known in the art, thus confirming predictability and expectation of success. Ex.1003, ¶82 (citing Ex.1007, Ex.1008). Any modification of de Castro's teachings, including implementing software programming or hardware, to accommodate the teachings of Robb, would have been within the level of ordinary skill in the art. Ex.1003, ¶83. For instance, a POSITA would have understood computer programming techniques and web design for publishing application services (e.g. Rayo protocol services) on a corporate portal. Ex.1003, ¶83.

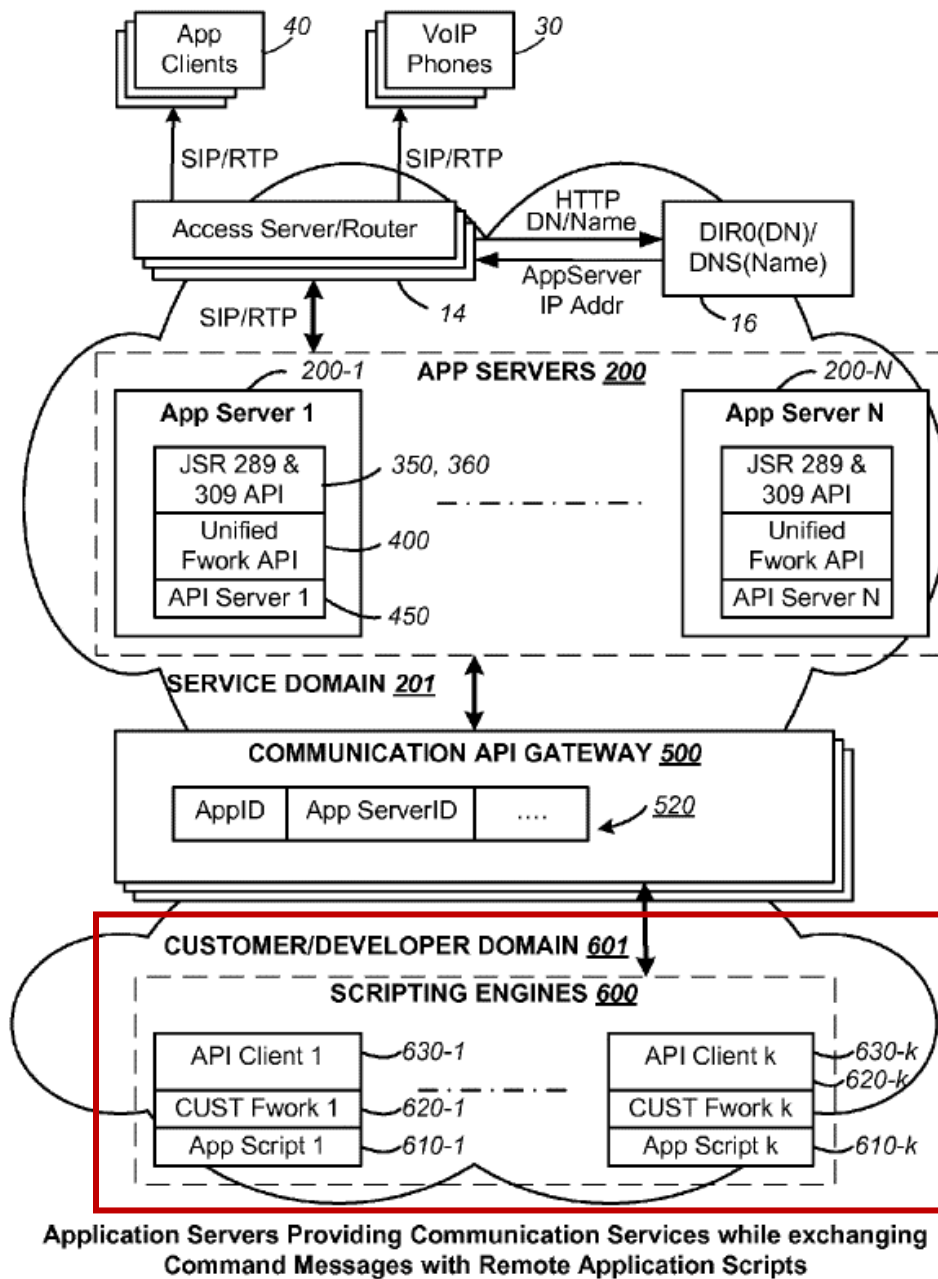
4. Claim 13

[13.0] *A method of developing and testing a new application service through a unified services platform, the method comprising:*

De Castro and Robb describe providing “a unified communication framework for programming communication applications” (a “*unified services platform*”). Ex.1005, [0070]-[0071]; *see also* [0086] (“unified framework programming model”), [0100] (“a unified framework for programming”).

De Castro teaches a “unified framework” (“*unified services platform*”) that “supports a unified API.” Ex.1005, [0038]-[0039]. The unified framework facilitates the deployment and execution of “application scripts” (“*provisioning a plurality of applications*”) for clients over a network. Ex.1005, Abstract, [0039], [0115]; Ex.1003, ¶¶84-85. De Castro explains in more detail that the unified framework is implemented to allow a “customer using the communication services [to] code the application script in a preferred programming language using a custom framework providing a set of customer-specific libraries.” Ex.1005, [0038], [0041]. The unified framework processes application scripts into “conformity to the API messaging protocol” using “embedded command[s] from the finite set of commands supported by the unified framework API 400.” Ex.1005, [0127]. Thus, De Castro describes allowing “application clients 40,” or end users, to access many applications through one unified framework despite the applications being scripted in potentially different programming languages. Ex.1005, [0038], [0041], [0115]; Ex.1003, ¶¶84-86.

In addition, de Castro describes how this unified framework is used by developers, described as customers (*see supra* section XI.C.1), to develop application scripts. In particular, Fig. 14, reproduced and annotated below, illustrates a “customer/developer domain 601.” Ex.1005, [0114], [0119], Fig. 14.



Ex.1005, Fig. 14 (annotated); Ex.1003, ¶87

The “customer/developer domain 601” includes “a plurality of scripting engines 600.” Ex.1005, [0119]. Customers “process ... application scripts 610” within the customer/developer domain 601. They “code the application script[s] in a preferred

programming language” using “a customer-specific framework 620 ... which supplies a set of customer-specific libraries.” Ex.1005, [0113], [0119]. Thus, de Castro describes “*a method of developing ... a new application service through a unified services platform.*” Ex.1003, ¶87.

Robb also describes “*developing and testing a new application service through a unified services platform.*” In particular, Robb describes an “application infrastructure platform (AIP)” that is used to provide application clients access to “multiple application services.” Ex.1006, Abstract. The AIP includes “multiple [application service provider] ASP systems providing application services to customers” (e.g., users) who are associated with telephone numbers. Ex.1006, Abstract, Fig.5. “The development environment provides the tools, component libraries, and Software Development Kits (SDKs) needed to rapidly build applications.” Ex.1006, 12:36-39, Fig. 21; Ex.1003, ¶88.

Robb describes a “Staging and Test Bed environment 503 [which] may be used to integrate new applications onto systems, stage them for scaled system network test, and finally prepare them for automatic software development to AIP sites.” Ex.1006, 17:5-9. Robb also describes how “[t]he AIP is associated with a development and test bed environment which conducts product and service development and testing.” Ex.1006, Abstract. Fig. 6A of Robb, reproduced below, illustrates the staging and test bed environment 503. Ex.1003, ¶89.

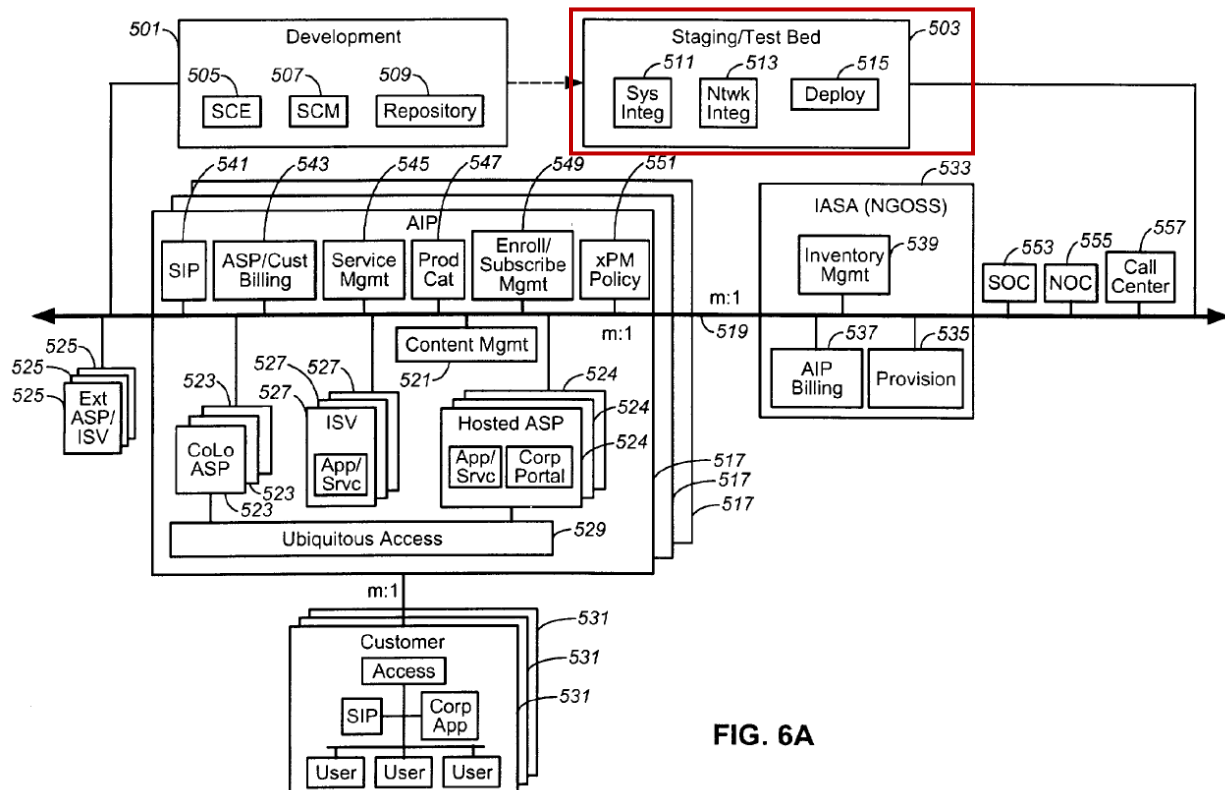


FIG. 6A

Ex.1006, Fig. 6A (annotated), Ex.1003, ¶89

Thus, to the extent the preamble is limiting, de Castro and Robb teach a “unified services platform” including a customer/developer domain for “developing and testing a new application service.” In addition, Robb describes an application infrastructure platform in which “integration and tests are executed.” Ex.1006, 31:22-23. This discloses the claimed “method.” Ex.1003, ¶90. As demonstrated below, the combination of de Castro and Robb discloses a “method comprising” the recited elements.

[13.1] establishing a plurality of service accounts for a corresponding plurality of application service provider systems;

First, de Castro teaches a plurality of “**scripting engines**” (“*plurality of application service provider systems*”) for “application scripts to be processed remotely.” Ex.1005, [0038], Fig.14; Ex.1003, ¶91.

Second, each scripting engine that provides application services is identified by an “**AppID**.” Ex.1005, Fig. 14. Information about these services is registered in a “**registry**.” Ex.1005, Fig.14, [0130]; Ex.1003, ¶92.

De Castro describes this registration process in more detail, including that “**registry 520...registers...the customers’ applications in the customer/developer domain.**” Ex.1005, [0130]. Within registry 520, also “logged are ... **the properties and status associated with each API clients 630.**” Ex.1005, [0130]. In particular, each scripting engine includes an API client 630, which “interprets [each App Script] and renders it in terms of the commands of the finite set” of “commands supported by the unified framework API 400.” Ex.1005, [0119], [0127]; Ex.1003, ¶93.

Fig.19 below illustrates registry 520 that logs information including the “properties/status” about each scripting engine’s API client 630 identified by an “App ID.” Ex.1005, Fig.19, [0130].

Registry 520 registers and logs information from each application service scripting engine (App ID) (“*plurality of service accounts*”) including application properties and status information

Registry 520 

SERVICE DOMAIN			USER DOMAIN		
App ServerID	App Server Properties/ Status		App ID	App Properties/ Status	

Ex.1005, Fig.19 (annotated); Ex.1003, ¶94

A POSITA would have understood that the information stored in the registry for each service is a “*service account[]*.” Ex.1003, ¶94. Registering each of the application services offered by the scripting engines renders obvious “*establishing a plurality of service accounts*.” Ex.1003, ¶¶94-95.

Thus, de Castro teaches a registry that registers and logs service-specific information (“*service accounts*”) about the scripting engine of each application service (“*application service provider systems*”), including logging application properties and status information, which renders obvious “*establishing a plurality of service accounts for a corresponding plurality of application service provider systems*.” Ex.1003, ¶96.

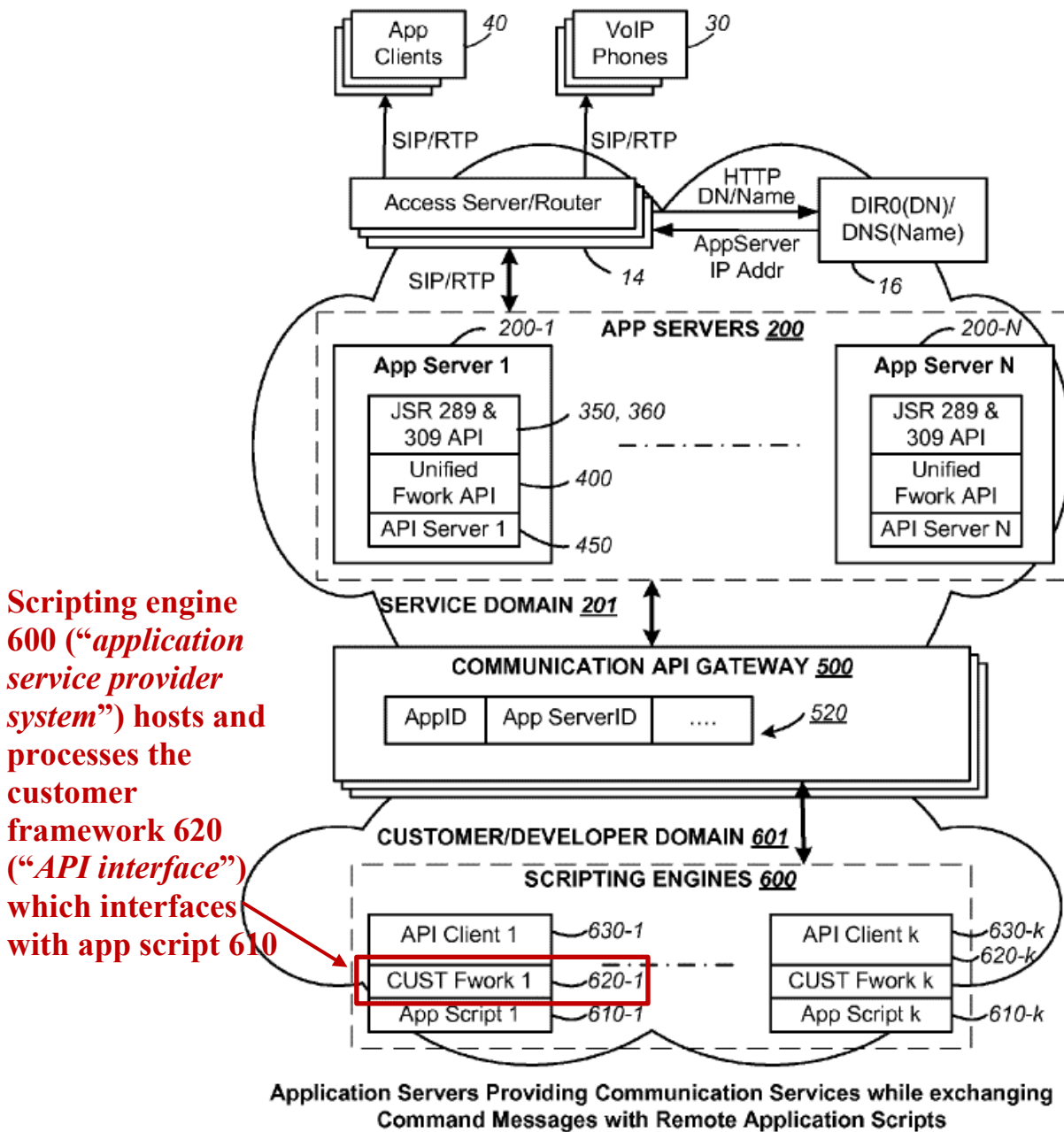
[13.2] receiving a plurality of API interfaces from the plurality of application service provider systems, wherein at least two of the API interfaces are different,

First, de Castro teaches that each scripting engine (“*application service*

provider systems”) includes a “**custom framework**” to interface with an application script. Ex.1005, [0119], [0067].

De Castro explains that a customer/developer “can code the application script in a preferred programming language using **a custom framework providing a set of customer-specific libraries.**” Ex.1005, [0041]. Each of these custom “frameworks” is an “API” (“*a plurality of API interfaces*”): “**Frameworks** are a special case of software libraries in that they are reusable abstractions of code wrapped in **a well-defined API.**” Ex.1005, [0067]. The custom framework API interfaces with the application script, as the “**processing of App Script 1 is facilitated by a customer-specific framework 620**, such as CUST Fwork 1 620-1 which supplies a set of customer-specific libraries.” Ex.1005, [0119]; Ex.1003, ¶¶97-98.

Fig.14 below illustrates scripting engine 600-1 hosting and processing the customer-specific framework 620-1 API, which interfaces with app script 610-1. Ex.1005, Fig.14, [0113].



Ex.1005, Fig.14 (annotated); Ex.1003, ¶99

Second, de Castro describes the application scripts and customer framework APIs as being “hosted” by the scripting engines. Ex.1005, [0039], [0110], [0115], [0119]. For instance, Fig.14 shown above illustrates customer framework APIs

(CUST Fwork 620-1 to 620-k) hosted in scripting engines (600-1 to 600-k).

Because the customer framework APIs are illustrated as being located within and hosted by the scripting engines, a POSITA would have understood that the customer framework APIs were “*receiv[ed]*” by the scripting engines. Ex.1003, ¶¶100-101. That is, scripting engines have customer frameworks implementing customer-specific APIs; and the scripting engines, at some point, must have “*receiv[ed]*” the customer frameworks. Ex.1003, ¶¶100-101.

De Castro also explains that application scripts and customer framework APIs are “**processed remotely** by appropriate scripting engines.” Ex.1005, Fig.14, [0038]. For instance, de Castro describes how a customer framework API facilitates processing by a scripting engine: “**processing of App Script 1 is facilitated by a customer-specific framework 620, such as CUST Fwork 1 620-1** which supplies a set of customer-specific libraries.” Ex.1005, [0119]. Within the scripting engine, “**processing is performed** by an API client 630 such as API client 1 630-1 which interprets App Script 1 and renders it in terms of the commands of the finite set defined at the API server 450 of the communication application server 200.” Ex.1005, [0119]. Therefore, the scripting engines’ hosting and processing of customer framework APIs renders obvious “*receiving a plurality of API interfaces.*” See Ex.1005, Fig.14, [0038], [0119]; Ex.1003, ¶102.

Third, de Castro teaches that its various customer framework APIs are

different (“*at least two of the API interfaces are different*”). Ex.1005, [0039]. As discussed above, the application script can be coded in a **“preferred programming language using a custom framework providing a set of customer-specific libraries.”** Ex.1005, [0041]. For instance, these languages “include Ruby, Python, Groovy and PHP.” Ex.1005, [0035]. Because the custom frameworks (“*API interfaces*”) are customer-specific and interface with applications written in different languages, it would have been obvious to a POSITA for the custom framework APIs to be different from one another. Ex.1005, [0067], [0041]; Ex.1003, ¶103.

Thus, de Castro teaches scripting engines hosting and processing customer framework APIs that interface with application scripts coded in different languages, which renders obvious “*receiving, by a processor system, a plurality of API interfaces from the plurality of application service provider systems, wherein at least two of the API interfaces are different.*” Ex.1003, ¶¶104-105.

[13.3] *integrating each of the plurality of API interfaces with a unified services platform system API interface;*

First, each “communication application server is provided with a **unified framework**” where “[t]he framework supports a unified API having class objects and functions conforming to a telephony object model.” Ex.1005, [0109], [0069]. In particular, the “**unified framework provides a unified API that gives**

access to a set of unified class objects for call control and media control.”

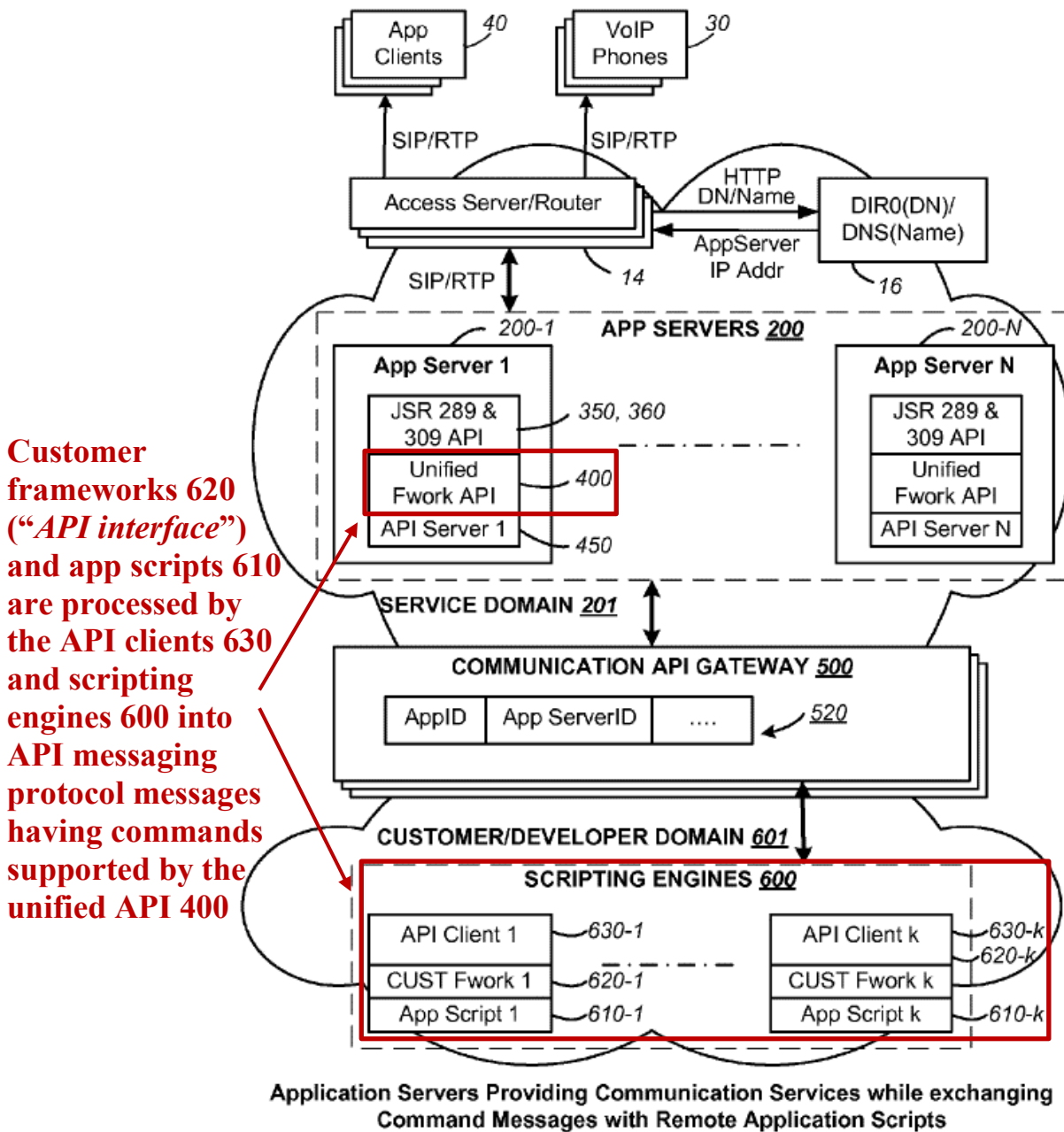
Ex.1005, [0100]. Therefore, “the communication application server 200 does not proceed to retrieve a script for the specific communication application,” but instead “it **exposes the unified communication framework API 400 as a finite set of commands.**” Ex.1005, [0118]. De Castro’s unified API is the claimed “*unified services platform system API interface.*” Ex.1003, ¶¶106-108.

Second, each of the scripting engines “interprets” or processes the customer frameworks (“*API interfaces*”) and application scripts, as it “interprets App Script 1 and renders it in terms of the commands of the finite set defined at the API server” supported by the unified API (“*integrating each of the plurality of API interfaces with a unified services platform system API interface*”). Ex.1005, [0119]; Ex.1003, ¶109.

For instance, scripting engines 600 include “API client 630 [which] **interprets App Script 1 and renders it in terms of the commands of the finite set defined at the API server.**” Ex.1005, [0119]. Next, the “API client 630 then **packages the commands as messages** in accordance with the messaging API protocol.” Ex.1005, [0119]. In this way, the application scripts and customer frameworks hosted by the scripting engines use “a messaging API protocol” to communicate with “the communication application server on the network to execute their applications as **a series of these commands**” expressed in the unified

API. Ex.1005, [0118]; Ex.1003, ¶110.

Fig.14 illustrates scripting engines 600 and API clients 630 that process the application scripts 610 and customer frameworks 620 into a finite set of commands supported by unified API 400. Ex.1005, Fig.14; Ex.1003, ¶111.



Ex.1005, Fig.14 (annotated); Ex.1003, ¶111

The scripting engines process application scripts into “conformity to the API messaging protocol” (“integrating”), which uses “embedded command[s] from the finite set of commands supported by the unified framework API 400”

(“*integrating with a unified services platform system API interface*”). Ex.1005, [0127]; Ex.1003, ¶112.

A POSITA would have found it obvious that de Castro’s system would “*integrat[e]*” the “*API interfaces*,” because integrating telecommunications services and their associated APIs was well known in the art. *See* Ex.1006, 3:31-39; Ex.1010, Fig. 2, 5:31-58; Ex.1012, 37:3-9, 458:5-22; Ex.1003 ¶113.

Thus, de Castro teaches scripting engines that process customer framework APIs into the commands of a unified API, which renders obvious “*integrating each of the plurality of API interfaces with a unified services platform system API interface*.” Ex.1003, ¶114.

[13.4] *receiving a definition of a service for each of the plurality of application service provider systems;*

De Castro’s “registry 520 [] **registers ... the customers’ applications** in the customer/developer domain 601.” Ex.1005, [0130]. Further, registry 520 also “**log[s] ... the properties and status associated with each API clients 630.**” Ex.1005, [0130]. De Castro thus teaches that the registry logs information for each application and the API client of a scripting engine (“*service provider systems*”). Ex.1005, [0130]. A POSITA would have understood that logging, at the registry 520, such information about the application scripts 610 and scripting engines 600, includes “*receiving*” and storing such information at the registry 520. Ex.1005,

[0130]; Ex.1003, ¶115.

Because the registry includes “properties” information for each scripting engine (Ex.1005, [0130]), a POSITA would have found it obvious for one of the “properties” to be a short description or name of the application (each an example “*definition of a service*”). Ex.1003, ¶116. Storing a service’s name, description, or both, would facilitate an administrator in determining to which service the other, related properties pertain. Ex.1003, ¶116. The registry also logs “status” information for each of the applications (Ex.1005, [0130]), which a POSITA would have found obvious to include service availability information (another example “*definition of a service*”). Ex.1003, ¶116.

Therefore, a POSITA would have understood that these teachings from de Castro render obvious “*receiving a definition of a service for each of the plurality of application service provider systems.*” Ex.1005, [0130]; Ex.1003, ¶117. As discussed above at Section IX, the ’048 patent describes a service name, description, and availability as examples of “*a definition of a service.*” Ex.1001, 15:65-6:8; Ex.1003, ¶117. De Castro’s disclosure of the registry including application properties (e.g., service name) and status information is just like the examples of service name, description, and availability in the ’048 patent, and further confirm that de Castro renders obvious this limitation. Ex.1005, [0130]; Ex.1003, ¶¶117-119.

Thus, de Castro teaches registering each service application in a registry, including logging properties and status information of each application's API client, which renders obvious "*receiving a definition of a service for each of the plurality of application service provider systems.*" Ex.1003, ¶120.

[13.5] *receiving one or more definitions of actions for each service for each of the plurality of application service provider systems, wherein the actions comprise responses to triggering events occurring for a telephone number;*

First, de Castro's system includes "**a finite set of commands**" ("*one or more definitions of actions*"), where "an application program essentially issues a series of such **commands to operate the communication application server.**" Ex.1005, [0109]; Ex.1003, ¶121. De Castro teaches "providing a messaging API protocol for customers of the communication application server on the network to **execute their applications as a series of these commands.**" Ex.1005, [0118]. Using this messaging protocol (e.g., Rayo protocol), "**a command carried in a message is received**" ("*receiving*") at the communication application server 200. Ex.1005, [0122]. That is, de Castro's system includes "[r]eceiving into the application server commands issued by the API client for invoking and manipulating the class objects relative to a call." Ex.1005, [0135]; Ex.1003, ¶121.

At the application communication server, "the API server **parses out the command in each message for execution in turn on the associate call actor.**" Ex.1005, [0124]. The parsed "commands" define the parameters of executable

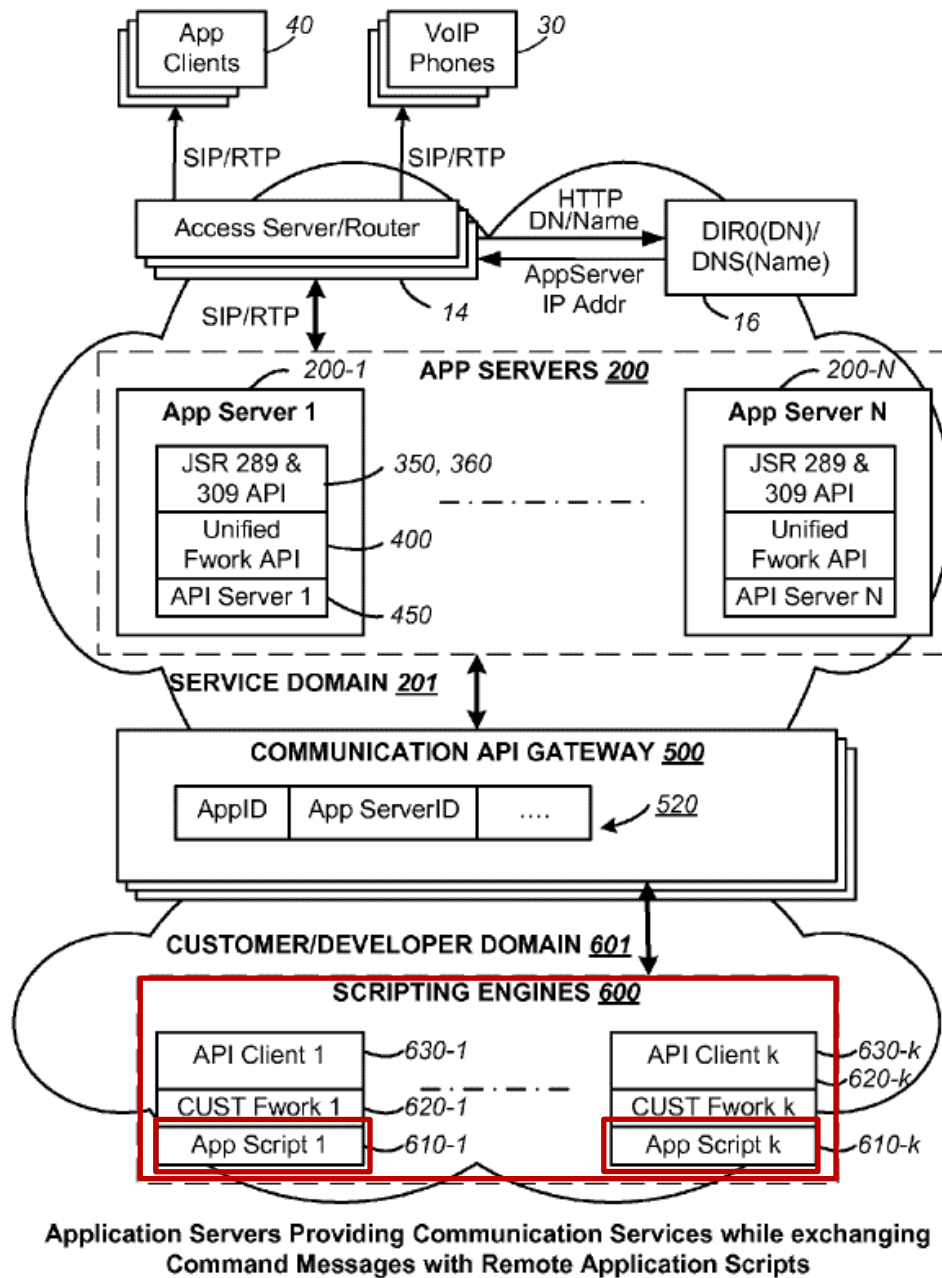
actions, such as to “**Answer**” or “**Redirect**” a telephone call. Ex.1005, [0141]; Ex.1003, ¶122.

Therefore, de Castro’s communication application server receives commands for a service, where the commands define telephone actions such as to answer or redirect a call, which teaches “*receiving one or more definitions of actions for each service.*” Ex.1003, ¶¶121-123.

Second, de Castro explains that a command in the “Rayo protocol primarily deals with calls” to a telephone number. Ex.1005, [0139]. For example, the command “**Redirect ... [o]nly applies to incoming calls.**” Ex.1005, [0141]. In an example, a call to the phone number “**to=‘tel:+18003211212’**” (“*triggering event occurring for a telephone number*”) can result in a redirect command “**redirect to=‘tel:+14152226789’**” (“*action*”). Ex.1005, [0141]. As another example, a call to a phone number where the “**[d]ial destination is busy and cannot answer the call**” (“*triggering events occurring for a telephone number*”) can result in an “**<end/> [event that] indicates that the call has been disconnected**” (“*actions*”). Ex.1005, [0142]; Ex.1003, ¶124.

Therefore, the “redirect” or “disconnect” commands that execute, in response to a call to or from a phone number, render obvious that “*the actions comprise responses to triggering events occurring for a telephone number.*” Ex.1005, [0141]-[0142]; Ex.1003, ¶124.

In addition, de Castro shows receiving definitions of actions “*for each service for each of the plurality of application service provider systems.*” Ex.1003, ¶125. As explained at [13.1], the scripting engines of de Castro correspond to *application service provider systems*. De Castro depicts scripting engines 600 in Fig. 14, shown below, and explains that the system includes multiple scripting engines 600. *See* Ex.1005, [0038] (“scripts [are] processed remotely by appropriate scripting engines.”); Ex.1003, ¶125.



Ex.1005, Fig.14 (annotated); Ex.1003, ¶125

In addition, each scripting engine 600 may include multiple application scripts, with the scripting engine 600 of Fig. 14 including k application scripts. See Ex.1005, Fig. 14. Each of these application scripts corresponds to *one or more*

definitions of actions in the form of a “finite set of commands.” Ex.1005, [0109]; *see also* Ex.1005, [0119] (for each app script of the scripting engines 600, the “API client 1 630-1 ... interprets App Script 1 and renders it in terms of the commands of the finite set defined at the API server 450.”); Ex.1003, ¶125. Thus, each app script (i.e., *service*) of each scripting engine (i.e., *application service provider system*) is rendered in terms of commands of the finite set, which correspond to *one or more definitions of actions*. Ex.1003, ¶125.

Thus, de Castro teaches a communication application server receiving Rayo messages including “answer” or “redirect” commands in response to a call to a phone number, which renders obvious “*receiving one or more definitions of actions for each service for each of the plurality of application service provider systems, wherein the actions comprise responses to triggering events occurring for a telephone number.*” Ex.1003, ¶126.

[13.6] *testing each triggering event and corresponding action for each service in a test environment to generate a response;*

As described at [13.0], de Castro describes the development of application scripts. *See e.g.*, Ex.1005, [0114], [0119]; Ex.1003, ¶127. De Castro also describes how developers “can code ... application script[s] in a preferred programming language using **a custom framework providing a set of customer-specific libraries.**” Ex.1005, [0041]; *see supra* [13.2]. The application scripts described in

de Castro specify “*triggering event[s]*”, such as inbound calls and outbound calls, and corresponding “*action[s]*”, such as redirecting an incoming call to a different number. *See* Ex.1005, [0141]-[0142]. While de Castro provides limited details of testing these triggering events and corresponding actions, testing triggering events and corresponding actions was well known in the art, as taught by Robb. Ex.1003, ¶¶127-128.

Robb teaches testing applications in a “Staging and Test Bed environment.” *See* Ex.1006, 17:5-9, Fig. 6A. Those applications have corresponding triggering events and actions implemented by a “Rules Engine.” *See* Ex.1006, 10:39-43, 10:52-55; *see also id.* 8:16-18. Because testing in Robb includes ensuring that new applications are properly integrated into the environment (*see* Ex.1006, 31:22-23 (the “Test Bed 2107” is “where **integration** and tests are executed”), it would have been obvious for Robb’s testing to include testing the triggering events and actions for new applications. Ex.1003, ¶129. Further details of Robb’s testing are provided below. Ex.1003, ¶129.

First, Robb describes testing new applications in a test environment. Fig. 6A (reproduced and annotated below) illustrates a “Staging/Test Bed” 503.

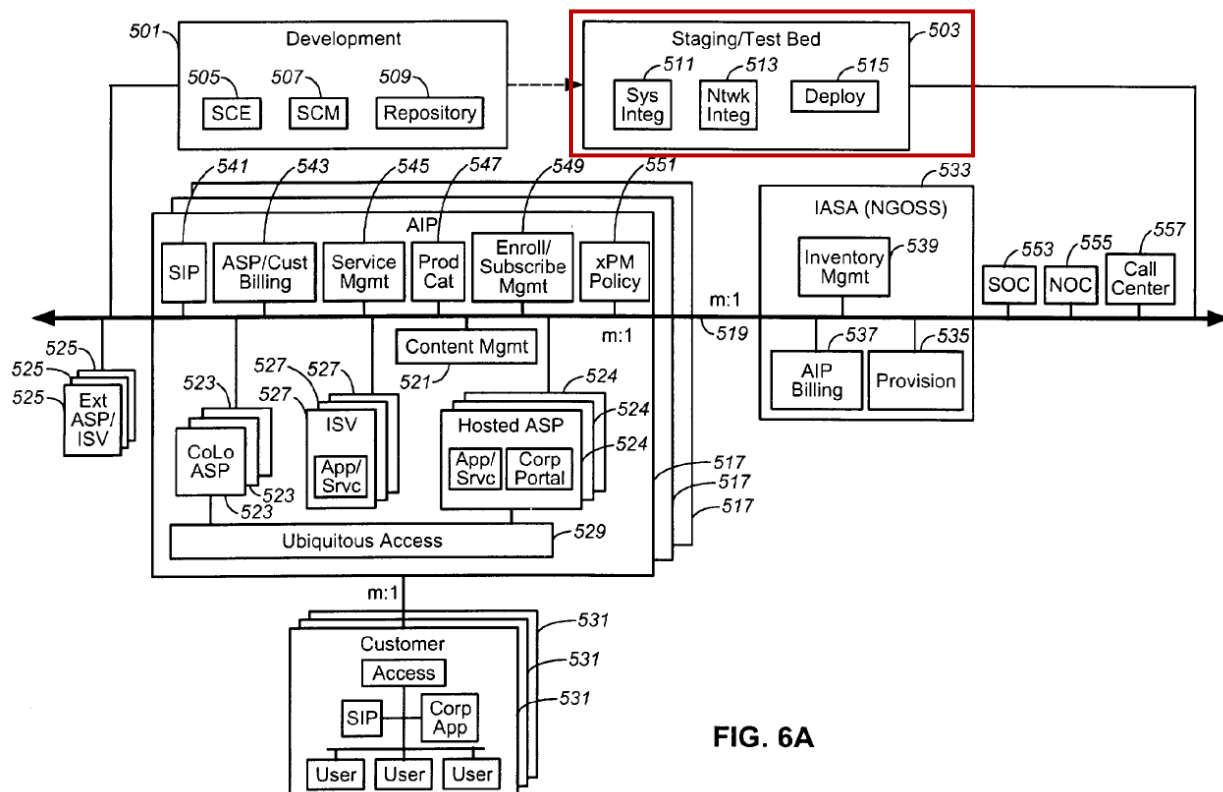


FIG. 6A

Ex.1006, Fig. 6A (annotated); Ex.1003, ¶130

“The Staging and Test Bed environment 503 may be used to **integrate new applications** onto systems, stage them for scaled **system network test**, and finally prepare them for automatic software deployment to AIP sites.” Ex.1006, 17:5-9.

The '048 patent explains that its claimed “*service[s]*” are provided to users via applications from application service providers. *See* Ex.1001, Fig. 4, 3:30-33, 6:1-3 (“provisioning a service from an application service provider”). Similarly, Robb describes “presenting services including application level services.” Ex.1006, 3:31-35; *see also* Ex.1006, 3:62-4:1 (“multiple [application service providers] provid[e]

application services to customer entities.”). Thus, Robb shows testing services in the staging and test bed environment 503. Ex.1003, ¶¶130-131.

Second, Robb explains how the applications tested in the staging and test bed environment are associated with triggering events and corresponding actions. Robb describes a “Rules Engine (RE)” that “evaluates **events** using a rule base and, when the criteria for a rule is met, fires the rule; rule firing can result in the **initiation of a specific work flow.**” Ex.1006, 10:39-43, 10:52-55; *see also id.* 8:16-18. Robb’s events correspond to the ’048 patent’s “*triggering event[s]*,” and the “specific work flow[s]” initiated in response to the events are “*corresponding action[s]*.” These disclosures in Robb are substantially similar to the ’048 patent’s description of a “policy and rules engine” that facilitates integrating multiple applications using *triggering event[s]* and *corresponding action[s]*. Ex.1001, 46:39-47:3; Ex.1003, ¶¶132-133.

Third, Robb teaches that testing a new application includes testing the integration of the new application, including testing the events and corresponding workflows for the new application. *See* Ex.1006, 8:41-48, 31:22-23, 32:65-67. Specifically, the AIP 114, which is “associated with” the “development and test bed environment” (Ex.1006, 4:21-25), “provides the core for ‘gluing’ services together” (Ex.1006, 8:45-48). The AIP 114 provides “a **tested** and integrated environment,” which shows that applications are tested prior to being put into the

AIP 114. Ex.1006, 8:41-48; *see also id.* 31:22-23 (the “Test Bed 2107” is “where **integration** and tests are executed.”). Then, Robb describes how further “verification tests” are performed to “prove that the AIP is fully functional.” Ex.1006, 32:65-67. Robb explains that application and service integration includes testing whether the triggering events and corresponding actions, which are implemented by the “Rules Engine,” are working properly. *See* Ex.1006, 7:65-8:3 (“The AIP 114 integrates applications and services together by binding them ... through the use of ... Rules Engines (REs).”). Similarly, “[t]he Staging and Test Bed environment 503 may be used to **integrate new applications** onto systems [and] stage them for scaled **system network test**.” Ex.1006, 17:5-9. In this way, Robb teaches not just testing of applications themselves, but also testing the entire “integrated environment” to ensure that triggering events and corresponding actions are performed properly to correctly integrate each new application into the ultimate package provided to the end user. *See* Ex.1006, 8:41-44 (“the AIP 114, may act as an ‘Application Infrastructure Provider’ by offering a **tested** and **integrated** environment.”), 7:38-41 (“The AIP 114 provides a Test Bed to integrate the solutions into a test AIP environment.”); Ex.1003, ¶134. Thus, Robb’s description of testing “new applications” in the Staging/Test Bed 503 and performing verification tests, including testing corresponding events and

workflows, renders obvious “*testing ... triggering event[s] and corresponding action[s] for ... service[s] in a test environment*” as claimed. Ex.1003, ¶¶134-135.

In addition, to the extent the term “*test environment*” was narrowed by the applicant during prosecution of the ’048 patent, a POSITA would have understood Robb’s description of testing as teaching the narrowed test environment as well. Ex.1003, ¶136.

In its response dated August 17, 2017, the applicant quoted the ’048 specification, explaining “[t]he test environment component 156 may respond in a manner that is the same or similar to the live implementation of the unified services platform 140 without allow [sic] the interactions to be released.” Ex.1002, 72 (quoting ’048 patent, 14:65-15:1). The applicant explained this passage to mean that “the test environment is the same as the unified services platform without the actual external connections or communications.” Ex.1002, 72; Ex.1003, ¶136. Based on these statements from the prosecution history, the claimed “*test environment*” may be interpreted as an environment that does not allow the interactions to be released because it is without the actual external connections or communications. As shown below, Robb’s description of testing teaches such a test environment. Ex.1003, ¶136.

A POSITA would have understood Robb’s test bed to similarly perform testing “without ... external connections or communications” in the same way the

applicant explained the claimed *test environment* to operate in the '048 patent because Robb's test bed is separate from the AIP and provides a test environment for integration testing. Ex.1002, 72; *see* Ex.1006, Fig. 6A, Fig. 8; Ex.1003, ¶137.

Robb illustrates the test bed as a separate environment from the live AIP environment, as illustrated in Fig. 6A, below.

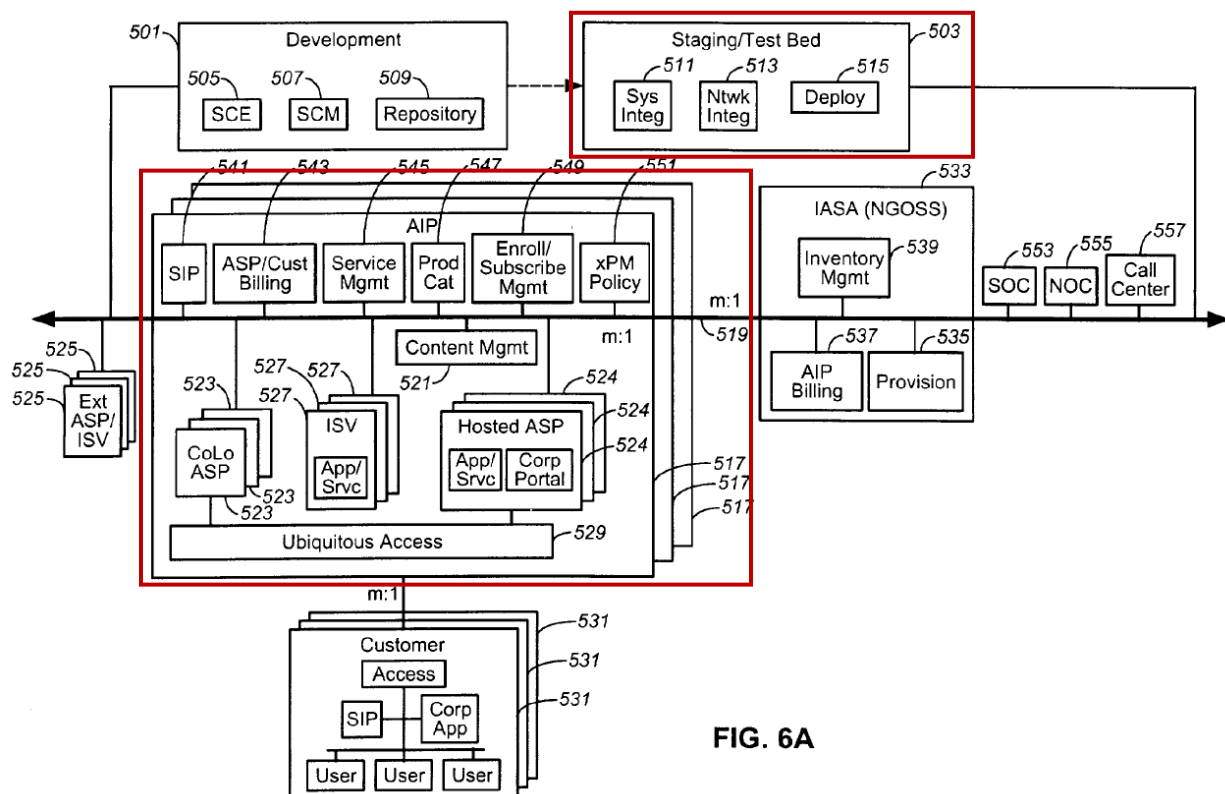
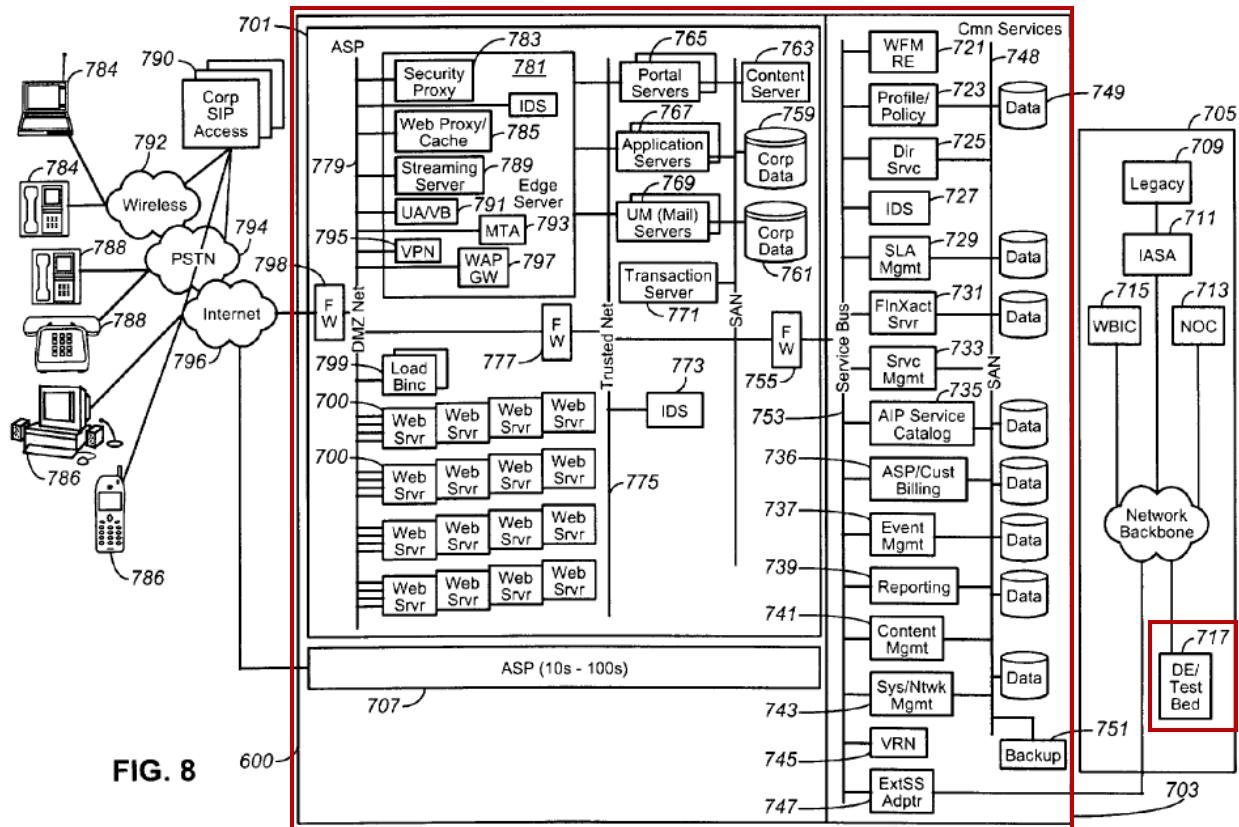


FIG. 6A

Ex.1006, Fig. 6A (annotated); Ex.1003, ¶137

As shown in Fig. 6A, the Staging/Test Bed 503 where applications are tested is separate from the AIP 517 where applications are deployed and run. In that regard, a POSITA would have understood that the Staging/Test Bed 503 can perform testing, including system integration and network integration without the need of

receiving actual inputs from AIP 517, or sending outputs to the AIP 517 (i.e., actual external communications or connections). This separation of the test environment and the live environment is also depicted in Fig. 8, shown below.



Ex.1006, Fig. 8 (annotated); Ex.1003, ¶138

Robb explains that “the Development and Test Bed 717” is “similar to” the development and test bed environments “shown at 501 and 503 in Fig. 6A.”

Ex.1006, 18:61-65. In Fig. 8, “the AIP is shown at [6]00.”⁴ Ex.1006, 18:58. Fig. 8 illustrates the test bed 717 within the proprietor environment 705, which is separate from the AIP 600. Ex.1003, ¶¶137-138.

A POSITA would also have understood that Robb’s test environment, in addition to being separate from the AIP, is “the same as” the AIP but “without the actual external connections or communications.” Ex.1002, 72. Specifically, Robb explains that “[t]he AIP 114 provides a Test Bed to integrate the solutions into a **test AIP environment** and for staging solutions and updates **for deployment to AIPs 114** over the Internet.” Ex.1006, 7:38-41. Testing integration of applications in a “test AIP environment” includes ensuring that the applications will work properly after deployment in the actual AIPs 114. *See* Ex.1006, 32:66-67 (integration tests “prove that the AIP is fully functional”). For this integration testing to be accurate and useful, a POSITA would have understood that the test AIP environment should functionally behave in the same way as the actual AIPs

⁴ Robb at 18:58 teaches that “the AIP is shown at 700.” However, the reference numerals 700 in Fig. 8 point to “web servers.” Elsewhere, Robb describes Fig. 8 including a “(DMZ) Net 779” connected to “Load Balancers 799 and a plurality of Web Servers 1100.” Ex.1006, 19:42-44. Therefore, “the AIP is shown at 700” is a typo and should read “the AIP is shown at 600.” Ex.1003, ¶138.

114. Ex.1006, 7:28-41; *see also* Ex.1003, ¶138. Thus, a POSITA would have understood that the “test AIP environment” of Robb, provided by the staging/test bed 503, would be “the same as” the actual AIP 114 but “without the actual external connections or communications” as explained by the applicant regarding the test environment of the ’048 patent. Ex.1002, 72; Ex.1003, ¶¶137-138.

The understanding that the staging/test bed 503 in Robb is without external communications is further supported by the prior art. Ex.1003, ¶¶139-140. For example, Desikan describes system testing.

[T]he live system is taken and **similar machines and configurations are mirrored**, and the operations from the users are **rerun on the mirrored deployment machine**.... Some deployments use intelligent recorders to record the transactions that happen on a live system and commit these operations on a mirrored system and then compare the results against the live system. **The object of the recorder is to help in keeping the mirrored and live system identical.**

Ex.1029, 136-37. Regarding integration testing, Desikan also describes the use of “stubs” to set up a test environment. *See* Ex.1029, 109 (“A stub simulates the interface by providing the appropriate values in the appropriate format **as would be provided by the actual component being integrated.**”). Similarly, McCaffrey describes performing “stub testing” or “mock testing.” Ex.1022, 91-93. McCaffrey gives an example of testing software used to make flight reservations in which a

stub system is created to avoid creating actual flight reservations in the live system. *See* Ex.1022, 91-93; Ex.1003, ¶139. Thus, a POSITA would have understood Robb’s “test AIP environment” (Ex.1006, 7:38-39) to teach a test environment that is a mirror of the live AIP environment but “without the actual external connections or communications” (Ex.1002, 72); Ex.1003, ¶¶139-140.

Robb also teaches testing the events (“triggering events”) and corresponding workflows (“corresponding actions”) for **each** service. In particular, Robb Fig. 28 describes a method whereby “an ASP add[s] service packs to a corporate portal.” Ex.1006, 34:16-17. Step 7 of the method includes running a verification test. *See* Ex.1006, Fig. 28. “In step 7, the ASP Administrator 2801 runs a sequence of **verification tests** and, together with the Corporate Administrator 2809, validates a selection of user roles.” Ex.1006, 34:29-31. Thus, each time a new service is provided by an ASP, the steps of the method of Fig. 28 are performed, including testing. Thus, Robb teaches “*testing ... triggering event[s] and corresponding action[s] for each service in a test environment*” as claimed. Ex.1003, ¶141.

To the extent that Robb does not specifically disclose testing “**each triggering event and corresponding action for each service**,” doing so would have been obvious to a POSITA at the time. Ex.1003, ¶142. As part of a POSITA’s bachelor’s degree education in electrical engineering, computer engineering, or computer science, a POSITA would have taken undergraduate courses in computer

programming and would have learned fundamental computer programming concepts, including the need to test their code. Ex.1003, ¶144. Application testing existed long before the priority date, and a POSITA would have known that an important aspect of testing a computer program was to test the software as comprehensively as possible. *See e.g.*, Ex.1022, 11 (“A full test pass suite generally consists of **every current test case.**”); Ex.1023, 7 (“complete testing” includes examining “**every aspect of the product.**”), 32 (“in function testing, you **test every function.**”), 35 (“coverage-based” testing includes “check[ing] that **every function behaves well** when used in combination with **any other function.**”), 37 (“[l]ogic testing attempts to check **every logical relationship** in the program.”), 38 (“[y]ou achieve 100 percent statement and branch coverage if you execute **every statement and every branch** from one statement to another.”). Ex.1003, ¶143. Therefore, a POSITA would have understood Robb’s description of testing and verifying each service as including “*testing each triggering event and corresponding action of each service.*” Ex.1003, ¶¶142-143.

In addition, a POSITA at the time would have understood that testing in Robb would include “*generat[ing] a response*” because testing commonly involved comparing a response from code-under-testing to an expected response. For example, those familiar with computer programming knew that “the most common activity in software testing is **sending input** to the system under test and

then **examining the result** to determine a pass or fail result.” Ex.1022, 5⁵. In addition, “[a] test case normally consists of **at least** the following items: a test case ID, a test case input, and **a test case expected result** or state.” Ex.1022, 5; *see also* Ex.1023, 14 (“testers just execute test cases and **compare what happens against expected results**”). Because the most common way of testing a piece of software was to compare its response to an expected response, a POSITA would have found it obvious for Robb’s testing processes to include having each triggered action “*generate a response*” that would then be compared to an expected response and used to evaluate whether the action was functioning correctly. Ex.1003, ¶145; *cf.* *Randall Mfg. v. Rea*, 733 F.3d 1355, 1363 (Fed. Cir. 2013) (obviousness may be shown by evidence of “a prevalent, perhaps even predominant, method of” doing something).

It would have been obvious to a POSITA to combine the teachings of de Castro and Robb. Ex.1003, ¶¶71-83. For example, it would have been obvious to a POSITA to test de Castro’s application scripts using a test environment, as taught by Robb, because comprehensively testing computer programs was known to be an important part of the software development process. Ex.1003, ¶77. A POSITA

⁵ Page 5 of Ex.1022 (McCaffrey) is not labelled because it is the first page of Chapter 1 as shown in the Table of Contents.

would have been motivated to apply Robb’s testing concepts to de Castro’s application scripts to ensure that the application scripts operated correctly.

Ex.1003, ¶77, 134. It was known that deploying a computer program that did not operate correctly was undesirable. Ex.1003, ¶¶137-140.

In summary, de Castro in combination with Robb teaches “*testing each triggering event and corresponding action for each service in a test environment to generate a response*” as claimed. Ex.1003, ¶146.

[13.7] *providing access to each service within a working environment after performing the testing; and*

Robb explains that services are deployed only after testing is complete. Referring to Fig. 6A, shown above, Robb describes “[t]he Staging and Test Bed environment 503 may be used to integrate new applications onto systems, stage them for scaled system network test and **finally** prepare them for **automatic software deployment to AIP sites.**” Ex.1006, 17:5-9. This description makes it clear that software deployment (“*providing access to each service*”) is done “finally,” or after the “scaled system network test.” Ex.1006, 17:5-9. After testing, a new application is moved from the Staging/Test Bed environment 503 to the AIP. *See* Ex.1006, 17:5-9. Robb also explains that the tests are run in the Test Bed 503 prior to “deployment to AIPs,” further showing that tests are completed prior to deployment. Ex.1006, 7:38-41; Ex.1003, ¶147.

Robb describes deploying software “to AIP Sites.” Ex.1006, 17:5-9. The AIP sites described in Robb are part of “the AIP platform” which Robb describes as an “integrated environment for **offering applications** from a network.” Ex.1006, 12:43-47. Thus, Robb’s “AIP Sites” and the “AIP platform” (separately or together) render obvious the claimed *working environment*. Ex.1006, 12:43-47, 17:5-9; Ex.1003, ¶147.

A POSITA would also have understood Robb’s disclosure regarding software testing and deployment as teaching deploying the software after running the tests, because this interpretation was consistent with software development and provisioning principles at the time. For instance, it was a known tenet of software development that “[f]ull test pass suites are run **before any milestone type of release** of the system under development, such as a beta release, an agile iteration release, or a major milestone (MM) release.” Ex.1022, 11. Similarly, in what is described as “one of the oldest [software development lifecycle] methodologies,” five phases are described in chronological order, including “requirements, design, implementation/coding, **verification/testing**, and **release/maintenance**.” Ex.1022,

35⁶. A POSITA would have been familiar with these basic concepts of computer programming from their undergraduate education. Ex.1003, ¶147.

Thus, de Castro in combination with Robb, teaches “*providing access to each service within a working environment after performing the testing,*” as claimed.

[13.8] *publishing, after testing, the definition of the service, the definition of the plan, and the one or more definitions of actions for each application service provider in a provisioning catalog.*

First, as discussed above at [13.4], de Castro teaches a registry that registers each application, including logging application “**properties and status**” information. Ex.1005, [0130]. And a POSITA would have found it obvious that such “properties and status” information for each application logged in a registry would include a name and short description of the application service. Ex.1005, [0130]; Ex.1003, ¶148.

De Castro also teaches “API client 630 such as API client 1 630-1 which **interprets App Script 1 and renders it in terms of the commands of the finite set** defined at the API server 450.” Ex.1005, [0119]. Therefore, the application services are defined by “commands” coded in an “application script” (“*definitions*

⁶ Page 35 of Ex.1022 (McCaffrey) is not labelled because it is the first page of Chapter 3 as shown in the Table of Contents.

of actions”). Ex.1005, [0119]; Ex.1003, ¶149.

Second, Robb, like de Castro, describes “providing application services to customers” who use “SIP[] phones.” Ex.1006, Abstract, 9:45-50. Robb further describes an application infrastructure platform (AIP) that “provides a collection of applications and services to the hosted ASPs 102; the ASPs 102 in turn may repackage and resell to their customers as corporate portals.” Ex.1006, 7:28-32. Specifically, Robb describes aggregating applications and services from ASPs and ISVs “into a **‘Product Catalog’**,” which “may be used by new ASPs to select their services for customization and resale to customers.” Ex.1006, 7:53-57, 17:62-67. Robb teaches that “[t]he aggregate product catalog enables service providers to leverage more customers by wholesaling or leasing to other service providers than they can address on their own.” Ex.1006, 26:47-49; Ex.1003, ¶¶150-156.

Robb teaches that the AIP also includes “**SLA [service-level agreement] Management**” which “provides for comparison of application or network events that affect delivery of applications and services to customers based on **SLA criteria**.” Ex.1006, 26:67-27:5. A service pack created or defined by the ASPs includes “an aggregation of services for a user role” and “SLA [service-level agreement] terms and conditions to the service.” Ex.1006, 34:24-28. Robb teaches that, once an ASP has selected a service to add to a corporate portal, an AIP administrator “specifies the details of the Service Level Agreements (SLA) agreed

by the AIP and the ASP.” Ex.1006, 33:39-42. The ’048 patent describes service limits associated with the service—such as SLA terms and conditions—as an example *definition of [a] plan*. Ex.1001, 16:5-8; Ex.1003, ¶150. Robb describes that an application infrastructure platform (AIP) may “aggregate its own applications and services with those of both external ASPs/ISVs and its own ASPs into an AIP’s **Product Catalog**.” Ex.1006, 3:17-20; Ex.1003, ¶155. “[C]ustomer ASPs may select from among numerous services in the AIP product catalog.” Ex.1006, 26:24-28; Ex.1003, ¶¶150-156.

It would have been obvious to a POSITA to apply Robb’s AIP product catalog (“*provisioning catalog*”) publishing technique to de Castro’s system. In the combination, information about de Castro’s available application scripts and associated commands (“*definitions of actions*”) would be published in a product catalog for service providers to aggregate, repackage, and resell. Ex.1003, ¶151. As suggested by Robb, the product catalog would also publish the related service properties (“*definition of the service*”) and default SLA terms (“*definition of the plan*”) associated with the service pack. Ex.1003, ¶152. A POSITA would have found this combination obvious at least for the reasons provided above and Section XI.C.3. Ex.1003, ¶153. It would have been obvious to a POSITA that the services added to the AIP’s product catalog would include default SLA terms, the “details” of which would then be “specifie[d]” by the AIP administrator. Ex.1006, 33:39-42,

32:40-42; Ex.1003, ¶153. For instance, the prior art acknowledges that a common practice was to provide default or standard terms for a software service agreement that could be customized or “superseded” by a service provider. Ex.1030, table at cols. 3 and 4 (billing and customer service (BCS) providers specify “default terms” of a subscription); Ex.1031, [0058] (providers publish “standard SLA templates” and “describe the details of the various SLA templates). This approach would simplify and streamline the service package publishing process by allowing ASPs to use the default SLA terms and modify as necessary, rather than define from the SLA terms and criteria from scratch. Ex.1031, [0058] (“To expedite and/or simplify SLA negotiation, consumers may retrieve standard SLA templates from the provider.”); Ex.1003, ¶152. Thus, de Castro in view of Robb renders obvious “publishing ... the definition of the service, the definition of the plan, and the one or more definitions of actions for each application service provider in a provisioning catalog.” Ex.1003, ¶¶150-156.

Third, as discussed above at [13.6], Robb teaches testing triggering events and corresponding actions with the “Staging and Test Bed Environment 503.” Ex.1006, 17:5-9. Robb uses the “Rules Engine (RE),” (Ex.1006, 10:43) to “evaluate[] events using a rule base” and “initiat[e] a specific work flow” (Ex.1006, 10:52-55), just as the ’048 patent uses a “policy and rules engine” to identify “triggering event[s]” and “invok[e] ... service[s] ... in response to the ...

triggering event[s]” (Ex.1001, 46:39-47:3). Ex.1003, ¶157.

As explained at [13.7], Robb makes clear that services are provided only after performing the test. In particular, “[t]he Staging and Test Bed environment 503 may be used to integrate new applications onto systems, stage them for scaled system network test, and **finally** prepare them for automatic software deployment to AIP sites.” Ex.1006, 17:5-9; Ex.1003, ¶158.

A POSITA would have found it obvious to combine Robb’s teachings regarding testing with de Castro’s system at least for the reasons provided above at element [13.4] and Section XI.C.3. Ex.1003, ¶159. In the resulting combination, when the access is provided to a service after testing (see 13.7 above), a service provider-accessible product catalog would also publish, for that application, a service pack including the application script (“*definitions of actions*”), application properties and status information such as the name and a short description of the service (“*definition of the service*”), and SLA terms and conditions (“*definition of the plan*”). Ex.1003, ¶159. Indeed, it would have been obvious for the service pack to include the app script to facilitate deploying the app script to a newly instantiated scripting engine when a user activates/subscribes to the service. Ex.1003, ¶159. Therefore, a POSITA would understand that de Castro in view of Robb renders obvious performing the “*publishing*” step “*after testing*.”

In summary, de Castro teaches a registry that logs properties and status

information for each application service, as well as an application script that includes executable commands. Robb teaches aggregating and publishing applications and services to a product catalog, and defining service packs having an aggregation of services and SLA terms and conditions. In combination, these teachings render obvious “*publishing, after testing, the definition of the service, the definition of the plan, and the one or more definitions of actions for each application service provider in a provisioning catalog.*” Ex.1003, ¶160.

5. Claim 14

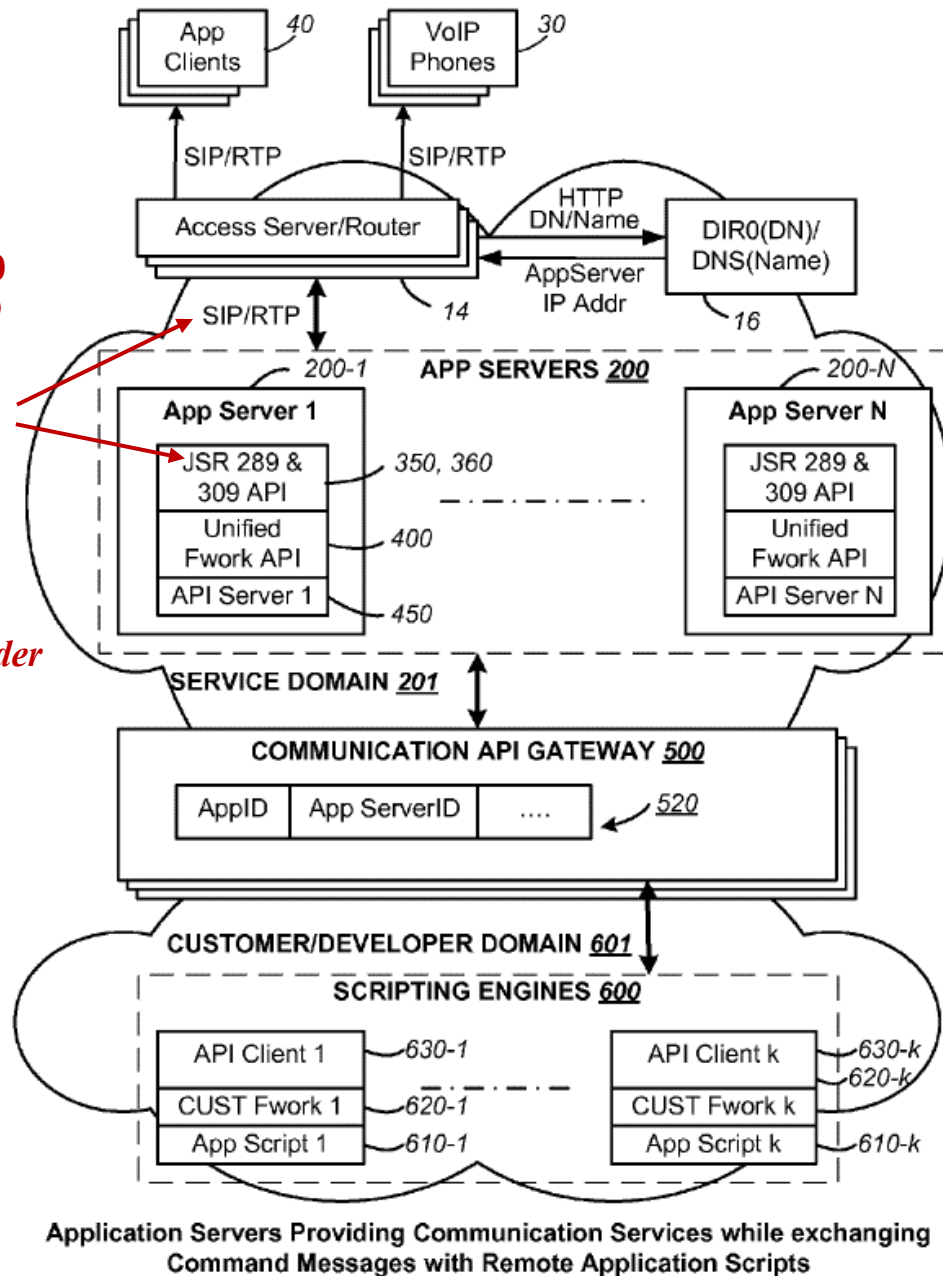
[14.1] *The method of claim 13, further comprising: receiving an API interface for a network provider system; and*

First, in de Castro, clients access the call-related application services “**via SIP**” networks (“*network provider system*”) connected to the application communication servers. Ex.1005, [0139], Fig.14; Ex.1003, ¶161.

Second, de Castro explains that the application communication servers 200 include a “**SIP Servlet (JSR 289) API 350**” (“*[a provider] API interface*”). Ex.1005, [0070]. Fig.14 below illustrates that SIP API (JSR 289) 350 executes on communication application server 200 for interfacing with a SIP network. Ex.1005, Fig.14, [0070], [0139]. Because the SIP API 350 executes on server 200, it would have been obvious to a POSITA for the SIP API to be loaded onto or “*receiv[ed]*” by server 200 prior to execution. *See* Ex.1005, Fig.14, [0070], [0139]; Ex.1003,

¶162. Figure 14 illustrates SIP API 350 as being included within, and thus already “receiv[ed]” by, server 200. Ex.1005, Fig.14.

App servers 200 receive JSR 289 API (e.g., SIP API) (“provider API interface”) for interfacing with a client-facing SIP network (“network provider svstem”)



Ex.1005, Fig.14 (annotated); Ex.1003, ¶162

Thus, de Castro teaches communication application servers including a SIP

API for interfacing with a SIP network, which renders obvious “*receiving [a provider] API interface for a network provider system.*” Ex.1003, ¶163.

[14.2] *integrating the API interface for the network provider system with the unified services platform system API interface, wherein at least one of the plurality of API interfaces from the plurality of application service provider systems is not compatible with the API interface for the network provider system.*

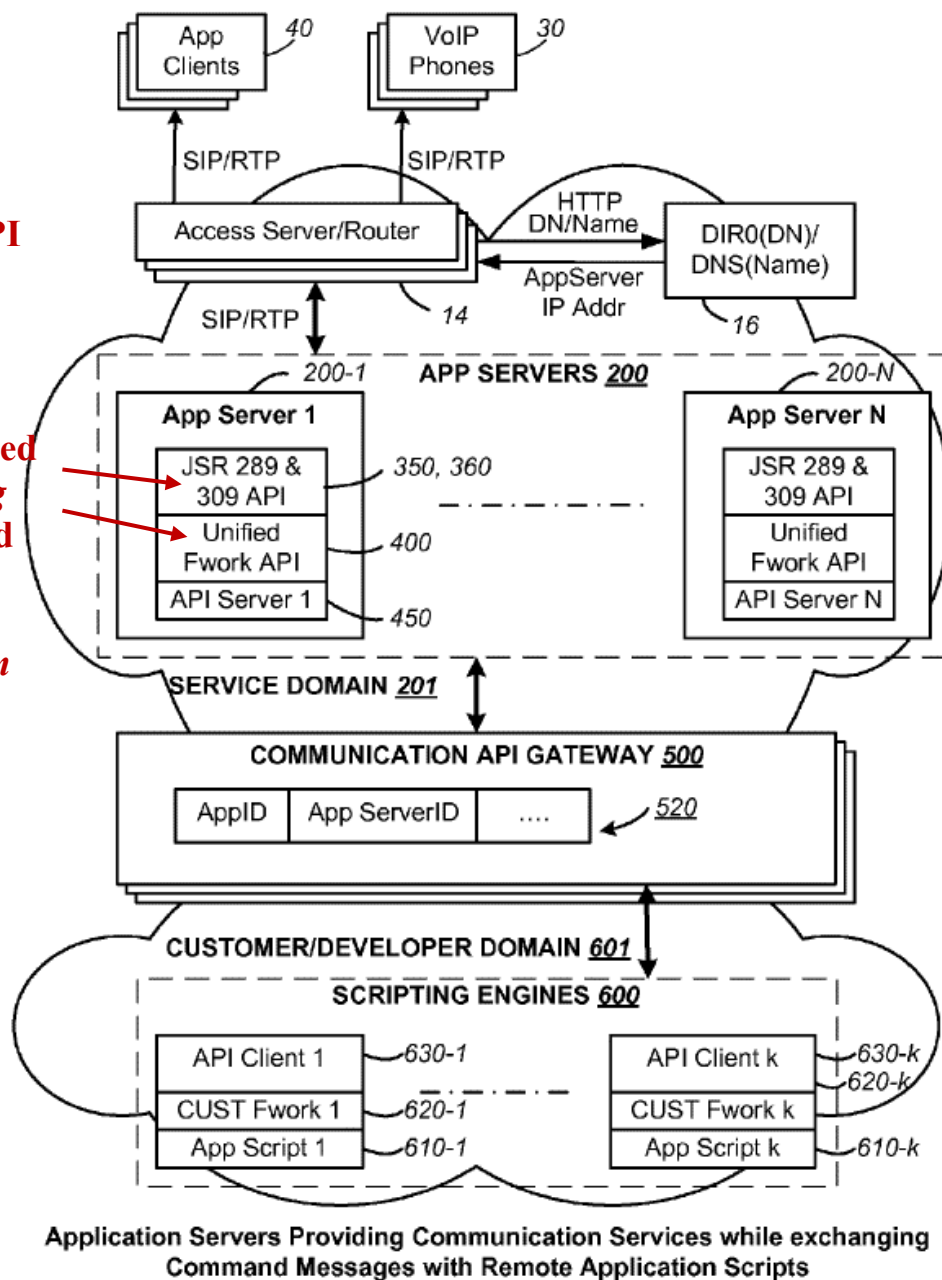
First, de Castro teaches that, within the communication application servers, the “JSR 289 API” (e.g., SIP API) is a programming interface that “deal[s] with high-level objects abstracted from low-level” SIP network communication objects, for use by the unified framework AIP 400. Ex.1005, [0070], [0074], Fig.14. Ex.1003, ¶164.

The “unified communication framework provides high-level, unified objects built from lower-level object of JSR 289/309”. Ex.1005, [0096]. Therefore, “[b]y dealing with **high-level objects abstracted from low-level objects of JSR 289**[], programming at the application level is more efficient and relevant to the problems at hand.” Ex.1005, [0074]; Ex.1003, ¶165.

Figure 14 depicts JSR 289 (SIP API) interfacing with unified framework API 400. Ex.1005, Fig.14. At this interface, JSR 289 abstracts and builds low-level call objects into high-level objects for use by unified framework API 400, which a POSITA would have understood to be the claimed “*integrating the provider API interface with the unified services platform system API interface.*” Ex.1005, Fig.14,

[0096]; Ex.1003, ¶166.

JSR 289 SIP API
(“*provider API interface*”) is
interfaced with
and provides
objects abstracted
for (“*integrating
... with*”) unified
framework API
400 (“*unified
services platform
API interface*”)



Ex.1005, Fig.14 (annotated); Ex.1003, ¶166

Second, de Castro’s SIP networks are associated with a “**SIP API**” or JSR 289 having low-level call objects. De Castro’s application services that utilize the

SIP networks use a **customer-specific framework** API that abstracts various programming languages such as Ruby or Python. *See* Ex.1005, [0067], [0070], [0041], [0119]; Ex.1003, ¶167.

For instance, “**SIP servlet (JSR 289) for call control ... provides a Java API**” (“*provider API interface*”), where Java is a type of programming language. Ex.1005, [0027]. “SIP servlets interact with clients by **responding to incoming SIP requests and returning corresponding SIP responses.**” Ex.1005, [0026]. The JSR 289 SIP API, therefore, deals with “**low level call control objects**” of an SIP network. Ex.1005, [0030], [0070]; Ex.1003, ¶168.

In contrast to JSR 289, which is an API for SIP network call objects, each of the customer-specific framework APIs 620 (“*API interfaces*”) are used for processing application scripts 610, which are written in different programming languages. Ex.1005, [0026], [0106]. De Castro explains that “application developers are using **other light-weight protocols and languages** to code the application scripts. These include **Ruby, Python, Groovy and PHP.**” Ex.1005, [0106]. Then, the application scripts, written in these various “**customer ... preferred programming language[s]**” are processed “**using a custom framework providing a set of customer-specific libraries.**” Ex.1005, [0106], [0041]; Ex.1003, ¶169.

Therefore, a POSITA would have understood that the Java-based JSR 289

SIP API is not compatible with the API of an application script coded in, e.g., Ruby, TAPI, or Python. Ex.1005, [0070], [0106]; Ex.1024, 461; Ex.1025, 723-727, 195, Ex.1003, ¶170. For instance, the SIP API is not natively compatible with the Rayo API, as Rayo messages cannot be parsed by a SIP server. Ex.1003, ¶¶170-172. These teachings from de Castro render obvious “*wherein at least one of the plurality of API interfaces from the plurality of application service provider systems is not compatible with the API interface.*” Ex.1003, ¶¶170-172.

Thus, de Castro teaches low-level call objects using a Java-based SIP API, which interfaces with and abstracts objects for a unified API, whereas application scripts coded in customer-preferred languages use customer-specific framework APIs, which renders obvious “*integrating the API interface with the unified services platform system API interface, wherein at least one of the plurality of API interfaces from the plurality of application service provider systems is not compatible with the API interface.*” Ex.1003, ¶173.

6. Claim 17

[17.0] *The method of claim 13, wherein the triggering event comprises a representational state transfer (REST) upon sending a message, a REST upon receiving a message, a REST before making a call, a REST after making a call, a REST before receiving a call, a REST after receiving a call, a Session Initiation Protocols (SIP) on receiving a call, a SIP on making a call, or any combination thereof.*

De Castro teaches routing inbound calls to a communication application,

which executes call control commands, “by looking up a directory service DIR0 16 with the IP address/dialed number in order to return the address of [an available] application [platform with the necessary resources to] execut[e] the specific communication application.” Ex.1005, [0116]. For the incoming call either to or from the client’s 40 telephone number, the directory DIR0 routes the call to/from the client and the server that executes the communication application, thus invoking the service. The server then can “control the call by using ... commands,” and the actions of the commands can be routed to the client 40. Ex.1005, [0141]; *see also id.* [0007] (discussing activation of various services, such as “Follow me, find me,” in response to inbound calls); Ex.1003, ¶174.

De Castro further describes the inbound call as using Session Initiation Protocols (SIP) (“*the triggering event comprises ... a Session Initiation Protocols (SIP) on receiving a call*”). Ex.1005, [0141]. De Castro explains that “[i]nbound calls originate from the PSTN or **via SIP.**” Ex.1005, [0139]. In an example Rayo protocol message for an Incoming Call service, the message includes code illustrating that receiving the incoming call includes using headers from the SIP protocol:

```
<!-- Signaling (e.g. SIP) Headers -->
<header name='Via' value='192.168.0.1' />
<header name='Contact' value='192.168.0.1' />
```

...

**<!-- Accept (e.g. SIP 180/Ringing). Only applies to
incoming calls. -->**

Ex.1005, [0141]; Ex.1003, ¶175.

De Castro further explains that “the call control API is the implementation of a standards-based SIP servlet container 340 and a **SIP servlet call control API 350**”—such that receiving a call includes using SIP for call control. Ex.1005, [0117], [0141]; Ex.1003, ¶176.

Thus, de Castro teaches receiving an incoming call including using SIP protocol to control receiving a call, which renders obvious “*wherein the triggering event comprises ... a Session Initiation Protocols (SIP) on receiving a call.*” Ex.1003, ¶177.

XII. DISCRETIONARY DENIAL IS NOT APPROPRIATE

A. No basis for 35 U.S.C. § 325(d) denial

Neither de Castro nor Robb were cited or considered by the Examiner during prosecution of the '048 patent.

Accordingly, discretionary denial under 35 U.S.C. § 325(d) is not appropriate because the first part of the Advanced Bionics framework is not met. *Advanced Bionics, LLC v. MED-EL Elektromedizinische Geräte GmbH*, IPR2019-01469, Paper 6 at 8 (PTAB Feb. 13, 2020) (precedential).

B. Discretionary denial under the *Fintiv* factors is not appropriate

The six factors for considering denial under § 314 strongly favor institution. *Apple Inc. v. Fintiv, Inc.*, IPR2020-00019, Paper 11 (Mar. 20, 2020) (precedential). The district court case is in its early stages. Patent Owner filed its complaint on May 20, 2022 (Ex.1020) and served Cisco on May 27, 2022. Patent Owner provided its preliminary infringement contentions on September 22, 2022. Ex.1019. Petitioner has since worked diligently to prepare this Petition, which is being filed well within the one-year timeframe allowed by Congress.

1. Whether evidence exists that a stay may be granted if a proceeding is instituted

A motion to stay has not yet been filed and the Board should not infer or attempt to predict the outcome if such a motion is later filed. *Sand Revolution II LLC v. Continental Intermodal Group – Trucking LLC*, IPR2019-01393, Paper 24 at 7 (June 16, 2020) (informative); *see also Dish Network L.L.C. v. Broadband iTV, Inc.*, IPR2020-01359, Paper 15 (Feb. 12, 2021) (“It would be improper to speculate, at this stage, what the Texas court might do regarding a motion to stay...”). Thus, this factor is neutral.

2. Proximity of the court’s trial date to the board’s projected statutory deadline for a final written decision

The operative Docket Control Order sets July 13, 2023, for the *Markman* hearing and January 22, 2024, for Jury Selection. Ex.1018, 3. The Final Written

Decision is expected in this case in September 2024. This factor nevertheless is neutral, because the Petitioner has worked expeditiously to prepare and file this petition just within six months of receiving Patent Owner’s infringement contentions on September 22, 2022. Ex.1019; *Fintiv*, Paper 11 at 11-12.

Furthermore, the current average time-to-trial for the district court hearing the co-pending case is 19 months. Ex.1017, 35; *In re Apple Inc.*, 979 F.3d 1332, 1350 (Fed. Cir. 2020). Thus, a trial in the co-pending case would be expected in December 2023. However, “scheduled trial dates are often subject to change.” *In re Apple Inc.*, 979 F.3d 1332, 1344, n.5 (Fed. Cir. 2020). Petitioner notes that four other cases are scheduled for Jury Selection on January 22, 2024—the same Jury Selection date as the co-pending litigation—and have not settled. Ex.1028 (listing all cases having Jury Selection scheduled for January 22, 2024).

As such, this factor is neutral. *See Sand Revolution II, LLC v. Continental Intermodal Grp.—Trucking LLC*, IPR2020-01393, Paper 24 (June 16, 2020) (panel granting institution on rehearing based on finding that uncertainty over district court’s trial date weighed against discretionary denial).

3. Investment in the parallel proceeding

The co-pending litigation is in its early stages, and the investment has been minimal. Fact discovery is not set to close until September 5, 2023, and expert discovery is not set to close until October 10, 2023. Ex.1018. The Docket Control

Order sets July 13, 2023 as the date for the Markman Hearing. Ex.1018, pp. 3.

Thus, this factor weights against discretionary denial. *See PEAG LLC v. Varta Microbattery GMBH*, IPR2020-01214, Paper 8 at 17 (Jan. 6, 2021) (finding that since no claim construction hearing had yet been held and discovery was not completed, the little investment in the parallel proceeding weighed against discretionary denial).

4. Overlap between issues raised in the petition and in the parallel proceeding

There will be no overlap of prior art issues. Petitioner stipulates that, if the Board institutes trial, Petitioner will cease asserting in the co-pending litigation the combination of references on which trial is instituted for the claims, to the extent Petitioner even asserts the same combination in the district court. This factor weighs against discretionary denial. *See Verizon v. Huawei*, IPR2020-01079, Paper 10 at 38 (Jan. 14, 2021) (finding a similar stipulation “mitigates the concern about overlapping issues” and weighs “against discretionary denial of the Petition.”).

5. Whether the petitioner is a defendant

Petitioner is a defendant in the litigation. Ex.1020. This is true of most Petitioners in IPR proceedings. While the *Fintiv* case indicates that a difference between the district court defendant and the petitioner may weigh against discretionary denial, nothing within the *Fintiv* case suggests that the same party

between the proceedings weighs in favor of discretionary denial. *See HP Inc. v. Slingshot Printing LLC*, IPR2020-01084, Paper 13 at 9 (having the “same parties as parallel proceeding” makes factor 5 “neutral”). Accordingly, this factor is neutral and should not be a basis for denying institution.

6. Other circumstances that impact the Board’s exercise of discretion, including the merits

As discussed above, the prior art presented in this Petition renders the Challenged Claims unpatentable as obvious. In view of the striking similarities between the ’048 patent claims and both de Castro and Robb, this petition “presents compelling evidence of unpatentability.” Director Vidal’s Guidance Memorandum, Ex.1021, 2. This factor weighs against discretionary denial.

As such, the *Fintiv* factors are either neutral or weigh against discretionary denial. Because this Petition was filed more than three months before the statutory bar date, institution should not be denied on discretionary factors.

C. Discretionary denial under *General Plastic* is not appropriate

The ’048 patent has not been challenged in any prior IPR petition, so no *General Plastic* discretionary institution factors apply here. *See General Plastic Indus. Co., Ltd. v. Canon Kabushiki Kaisha*, IPR2016-01357, Paper 19 at 16 (Sept. 6, 2016) (Section II.B.4.i. precedential).

XIII. CONCLUSION

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

Respectfully submitted,

Dated: April 4, 2023
HAYNES AND BOONE, LLP
2323 Victory Avenue, Suite 700
Dallas, Texas 75219
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 '048 patent is or was involved in the following cases:

Case Heading	Number	Court	Filed
<i>Zilkr Cloud Technologies, LLC v. Cisco Systems, Inc.</i>	2-22-cv-00166	EDTX	May 20, 2022
<i>Zilkr Cloud Technologies, LLC v. RingCentral, Inc.</i>	3-21-cv-02807	NDTX	Nov. 11, 2021

Additionally, the '048 patent is related to U.S. Patent No. 9,210,254, which is involved in the *Inter Partes* Review proceeding IPR2023-00553. The '048 patent is also related to U.S. Patent No. 9,998,607, which is involved in the *Inter Partes* Review proceeding IPR2023-00663.

C. Lead and back-up counsel and service information

Lead Counsel

Theodore M. Foster
HAYNES AND BOONE, LLP
2323 Victory Ave. Suite 700
Dallas, TX 75219

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

Back-up Counsel

Samuel Drezdzon
HAYNES AND BOONE, LLP
2323 Victory Ave. Suite 700
Dallas, TX 75219

Phone: (972) 739-6918
Fax: (214) 200-0853
samuel.drezdzon.ipr@haynesboone.com
USPTO Reg. No. 67,085

Brett Bostrom
HAYNES AND BOONE, LLP
2323 Victory Ave. Suite 700
Dallas, TX 75219

Phone: (972) 739-6973
Fax: (214) 200-0853
brett.bostrom.ipr@haynesboone.com
USPTO Reg. No. 73,009

Eugene Goryunov
HAYNES AND BOONE, LLP
2323 Victory Ave. Suite 700
Dallas, TX 75219

Phone: (312) 216-1630
Fax: (214) 200-0853
eugene.goryunov.ipr@haynesboone.com
USPTO Reg. No. 61,579

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.

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 13,066. 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: April 4, 2023

/Theodore M. Foster/

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

CERTIFICATE OF SERVICE

The undersigned certifies that, in accordance with 37 C.F.R. § 42.6(e) and 37 C.F.R. § 42.105, service was made on Patent Owner as detailed below.

Date of service April 4, 2023

Manner of service FEDERAL EXPRESS

Documents served Petition for *Inter Partes* Review Under 35 U.S.C. § 312
and 37 C.F.R. § 42.104 of U.S. 9,883,048;
Petitioner's Exhibit List;
Exhibits 1001-1031

Persons served Conley Rose, P.C.
4965 Preston Park Blvd, Suite 195E
Plano, TX 75093

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