

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

**DECLARATION OF HENRY HOUH, PH.D.,
UNDER 37 C.F.R. § 1.68 IN SUPPORT OF PETITION
FOR *INTER PARTES* REVIEW**

TABLE OF CONTENTS

I.	INTRODUCTION	3
II.	QUALIFICATIONS AND PROFESSIONAL EXPERIENCE	6
III.	LEVEL OF ORDINARY SKILL IN THE ART	11
IV.	RELEVANT LEGAL STANDARDS	12
V.	BACKGROUND	14
A.	Telecommunication Application Services	14
B.	Application Programing Interfaces (APIs).....	16
VI.	OVERVIEW OF THE '048 PATENT	18
VII.	CLAIM CONSTRUCTION	22
VIII.	IDENTIFICATION OF HOW THE CLAIMS ARE UNPATENABLE	23
IX.	THE COMBINATION OF DE CASTRO AND ROBB RENDERS OBVIOUS CLAIMS 13, 14, AND 17.....	23
A.	Summary of de Castro	23
B.	Summary of Robb	27
C.	REASONS TO COMBINE DE CASTRO WITH ROBB	31
D.	Detailed Analysis of Claims.....	38
1.	Claim 13	38
2.	Claim 14	85
3.	Claim 17	92
X.	CONCLUSION.....	95

I, Henry Houh, Ph.D., do hereby declare as follows:

I. INTRODUCTION

1. I am making this declaration at the request of Cisco Systems, Inc. in the matter of the *Inter Partes* Review of U.S. Patent No. 9,883,048 (“the ’048 patent”) to Hunt et al.

2. I am being compensated for my work in this matter at my standard hourly rate. I am also being reimbursed for reasonable and customary expenses associated with my work and testimony in this proceeding. My compensation is not contingent on the outcome of this matter or the specifics of my testimony.

3. I have been asked to provide my opinions regarding whether the subject matter of claims 13, 14, and 17 (“the Challenged Claims”) of the ’048 patent would have been obvious to a person having ordinary skill in the art (“POSITA”) at the time of the alleged invention, in light of the prior art. It is my opinion that the Challenged Claims would have been obvious to a POSITA.

4. In the preparation of this declaration, I have studied:

- a. the ’048 patent, Ex.1001.
- b. the prosecution history of the ’048 patent (“’048 File History”), Ex.1002.
- c. U.S. Pat. Pub. No. 2012/0016932 to de Castro et al. (“de Castro” or “De Castro”), Ex.1005.

d. U.S. Pat. No. 8,126,722 to Robb (“Robb”), Ex.1006.

5. In forming the opinions expressed below, I have considered: the documents listed above; the relevant legal standards, including the standard for obviousness; and my own knowledge and experience based upon my work in the field of network communications and security as described below, as well as portions of the following additional materials:

a. “API Reference – Developers,” Zapier (archived January 23, 2013) (“Zapier 1”), Ex.1007.

b. “Integrate your Web Services,” Zapier (archived January 23, 2013) (“Zapier 2”), Ex.1008.

c. Ben Langfeld, Jose de Castro, “XEP-0327: Rayo,” XMPP Standards Foundation (archived August 18, 2013) (“the Rayo Specification”), Ex.1009.

d. U.S. Pat. No. 7,593,980 to Marascio et al. (“Marascio”), Ex.1010.

e. U.S. Pat. No. 7,496,054 to Taylor (“Taylor”), Ex.1011.

f. U.S. Pat. No. 9,557,889 to Raleigh et al. (“Raleigh”), Ex.1012.

g. U.S. Pat. No. 8,600,836 to Collins, (“Collins”) Ex.1030.

h. U.S. Pat. Pub. No. 2007/0033194 to Srinivas et al. (“Srinivas”) Ex.1031.

i. James McCaffrey, “Software Testing: Fundamental Principles and Essential Knowledge” (2009) (“McCaffrey”) Ex.1022.

j. Cem Kaner, James Back, Bret Pettichord, “Lessons Learned in Software Testing: A Context-Driven Approach” (2002) (“Kaner”) Ex.1023.

k. Srinivasan Desikan, Gopalaswamy Ramesh, “Software Testing: Principles and Practices” (2011) (“Desikan”) Ex.1029.

l. Microsoft Press Computer Dictionary (3d ed. 1997) (“Microsoft”) Ex.1024.

m. Newton’s Telecom Dictionary (18th ed. 2002) (“Newton”) Ex.1025.

6. I understand that prior art includes printed publications which were publicly available prior to the earliest priority date of the ’048 patent. I have also considered the following documents as demonstrating the prior art status of the non-patent literature documents listed above, including the Rayo Specification, Zapier 1, and Zapier 2:

- Affidavit of Authenticity of “XEP-0327: Rayo,” confirming the prior art status of the Rayo Specification, Ex.1016;
- Declaration of Jose de Castro under 37 C.F.R. § 1.68, confirming the prior art status of the Rayo Specification, Ex.1013; and
- Affidavit of Authenticity of Zapier 1 and Zapier 2, confirming the prior art status of Zapier 1 and Zapier 2, Ex.1014 and Ex.1015.

7. Unless otherwise noted, all **emphasis** in any quoted material has been added. Claim terms are *italicized*.

II. QUALIFICATIONS AND PROFESSIONAL EXPERIENCE

8. My complete qualifications and professional experience are described in my *Curriculum Vitae*, a copy of which can be found in Ex.1004. The following is a brief summary of my relevant qualifications and professional experience.

9. I received a Ph.D. in Electrical Engineering and Computer Science from the Massachusetts Institute of Technology (“MIT”) in 1998. Beforehand, I received a Master of Science degree in Electrical Engineering and Computer Science in 1991, a Bachelor of Science degree in Electrical Engineering and Computer Science in 1989, and a Bachelor of Science degree in Physics in 1990, all from MIT.

10. I am currently self-employed as an independent technical consultant. I am also president of a company that provides supplemental science, technology, engineering, and mathematics (“STEM”) education to children of all ages.

11. I first entered telecommunications in 1987 when I worked as a summer intern at AT&T Bell Laboratories as part of a five-year dual degree program at MIT. I continued to work at AT&T Bell Laboratories as part of this MIT program. While at MIT, I was a teaching assistant (“TA”) in the Electrical Engineering and Computer Science Department’s core Computer Architectures

course. I first was a TA as a senior for a role typically reserved for graduate students. I later became head TA. The course covered various topics in computer architectures. As a TA, I helped write homework assignments, lab assignments, and exams. I also taught in the recitation sections.

12. Later, as part of my doctoral research at MIT from 1991-1998, I was a research assistant in the Telemedia Network Systems (“TNS”) group at the Laboratory for Computer Science. The TNS group built a high-speed gigabit network and created applications that ran over the network. Example applications included ones for remote video capture, processing, and display of video on computer terminals. In addition to working on the design of core network components, designing and building the high-speed links, and designing and writing the device drivers for the interface cards, I also set up the group’s web server.

13. I also helped to build the web pages that initiated the above-mentioned video sessions via a web interface. Vice President Al Gore visited our group in 1996 and received a demonstration of—and remotely drove—a radio-controlled toy car with a wireless video camera mounted on it that was built by our group. This toy car device received commands transmitted over a network from a remote computer, and video data from the toy car was transmitted wirelessly then over a computer network back to the user controller. On occasion, we allowed

users visiting our web site to drive the toy car from their remote computer while they watched the video on their computer. The video stream was encoded by TNS-designed hardware, streamed over the TNS-designed network, and displayed using TNS-designed software.

14. I defended and submitted my Ph.D. thesis, titled “Designing Networks for Tomorrow’s Traffic,” in January 1998. As part of my thesis research, I analyzed local area and wide area flows to show a more efficient method for routing packets in a network, based on traffic patterns at the time.

15. From 1997 to 1999, I was a Senior Scientist and Engineer at NBX Corporation, a start-up that made business telephone systems for streaming packetized audio over data networks instead of using traditional telephone lines. NBX was later acquired by 3Com Corporation and the phone system is still used today by numerous businesses.

16. As part of my work at NBX, I designed the core audio reconstruction algorithms for the telephones, as well as the packet transmission algorithms. I also designed and validated the core packet transport protocol used by the phone system. The protocol was used for all signaling in the phone system, including for the setup of conference calls.

17. The NBX system also featured a computer interface for initiating phone calls, which could also initiate conference calls. The NBX system also

supported the Telephony Application Programming Interface (“TAPI”) that allowed other computer programs to integrate with our system telephony features. We obtained U.S. Patent No. 6,697,963, entitled “Telecommunication method for ensuring on-time delivery of packets containing time-sensitive data,” as part of this work.

18. From 1999-2004, I was employed by Empirix or its predecessor company, Teradyne. Empirix was a leader in test tools for software and telecommunications protocols and systems, providing functional testing tools as well as load testing tools. From 2000-2001, I conceived and built a test platform for testing Voice-over-IP (VoIP). The first application on this new test platform was a cloud emulator for simulating the effects of transmitting VoIP over a busy network. Following this project, I became the architect for a next-generation software web testing platform developed by Empirix, known as e-Test Suite. e-Test Suite is now owned by Oracle Corporation. e-Test provided functional and load testing for web sites. e-Test emulated a user’s interaction with a web site and provided web developers with a method of creating various scripts and providing both functional testing (i.e., did the web site provide the correct response) and load testing (i.e., could the web site handle 5000 users simultaneously). As part of this project, I co-wrote a javascript interpreter for e-Test so that it could understand the javascript code present on advanced web sites. During this time, I was invited

several times to present a guest lecture on software testing for an upper-level undergraduate software development class at MIT.

19. In 2006, as part of my role at BBN Technologies, I helped found PodZinger Inc., now known as RAMP Inc. PodZinger utilized BBN's speech recognition algorithms to search through the spoken words in audio and video segments. While I was Vice President of Operations and Technology, PodZinger followed its initial prototype with a full streaming audio and video search solution. I also created a social networking web site, which BBN sold to a venture-funded startup company. In the process of creating the web site, I designed and specified the authentication and authorization protocols.

20. I have been awarded several United States patents, and I have several patent applications pending including the following examples:

21. U.S. Patent No. 7,975,296, "Automated security threat testing of web pages";

22. U.S. Patent No. 7,877,736, "Computer language interpretation and optimization for server testing";

23. U.S. Patent No. 7,801,910, "Method and apparatus for timed tagging of media content";

24. U.S. Patent 7,590,542, "Method of Generating Test Scripts Using a Voice-Capable Markup Language";

25. U.S. Patent No. 6,967,963, “Telecommunication method for ensuring on-time delivery of packets containing time-sensitive data”;

26. U.S. Patent Application Publication No. 20070106685, “Method and apparatus for updating speech recognition databases and reindexing audio and video content using the same”;

27. U.S. Patent Application Publication No. 20070106693, “Methods and apparatus for providing virtual media channels based on media search”;

28. U.S. Patent Application Publication No. 20070106760, “Methods and apparatus for dynamic presentation of advertising, factual, and informational content using enhanced metadata in search-driven media applications”;

29. U.S. Patent Application Publication No. 20070112837, “Method and apparatus for timed tagging of media content”;

30. U.S. Patent Application Publication No. 20070118873, “Methods and apparatus for merging media content”; and

31. U.S. Patent Application Publication No. 20090222442, “User-directed navigation of multimedia search results.”

III. LEVEL OF ORDINARY SKILL IN THE ART

32. I understand there are multiple factors relevant to determining the level of ordinary skill in the pertinent art, including (1) the levels of education and experience of persons working in the field at the time of the invention; (2) the

sophistication of the technology; (3) the types of problems encountered in the field; and (4) the prior art solutions to those problems.

33. 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 or graduate degree in 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.

34. For purposes of this Declaration, in general, and unless otherwise noted, my statements and opinions, such as those regarding my own experience and what a POSITA would have understood or known generally (and specifically related to the references I consulted herein), reflect the knowledge that existed in the relevant field as of the priority date of the ’048 patent.

IV. RELEVANT LEGAL STANDARDS

35. I am not an attorney. In preparing and expressing my opinions and considering the subject matter of the ’048 patent, I am relying on certain basic legal principles that Cisco’s counsel has explained to me.

36. I understand that prior art to the ’048 patent includes patents and printed publications in the relevant art that predate the priority date of the ’048

patent. For purposes of this Declaration, I am applying October 9, 2013, as the priority date of the '048 patent.

37. I have been informed by Cisco's counsel that a claimed invention is unpatentable under 35 U.S.C. § 103 if the differences between the claimed invention and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a POSITA. I have also been informed by Cisco's counsel that the obviousness analysis considers factual inquiries, including the level of ordinary skill in the art, the scope and content of the prior art, and the differences between the prior art and the claimed subject matter.

38. I have been further informed by Cisco's counsel that there are several recognized rationales for combining references or modifying a reference to show obviousness. These rationales include: (a) combining prior art elements according to known methods to yield predictable results; (b) simple substitution of one known element for another to obtain predictable results; (c) use of a known technique to improve a similar device (method, or product) in the same way; (d) applying a known technique to a known device (method, or product) ready for improvement to yield predictable results; (e) choosing from a finite number of identified, predictable solutions, with a reasonable expectation of success; and (f) some teaching, suggestion, or motivation in the prior art that would have led a

POSITA to modify the prior art or to combine prior art teachings to arrive at the claimed invention.

V. BACKGROUND

A. Telecommunication Application Services

39. It was and still is common for people to subscribe to a telecommunication network provider for cellular service or land-line phone service. A service subscriber gets a telephone number at which other subscribers can reach them, and calls are connected over one or more networks, depending on where the subscribers are located. De Castro, [0007].

40. For example, if the subscribers are both employees in the same office building, which includes wiring and connections for a private telephone network, then only the private network may be used to connect the phone call. Or, if one subscriber is in the office and the other subscriber uses a cell phone remotely, the call may be connected over: the private office network, which in turn connects to the traditional landline phone system also known as the public switched telephone network (PSTN), which lastly connects to a cellular network operated by a carrier (AT&T, Verizon, etc.). Robb, 10:36-38, 12:12-15, 14:40-54.

41. Phone calls could also use an internet protocol (IP) network where one or both subscribers uses a voice-over-IP (VoIP) phone. One common type of VoIP network was, and still is today, session initiation protocol (SIP), which

supports the basic call processing functions and features used in the PSTN. Robb, 9:43-55, 10:25-38. An internet network carrying SIP protocol data may be referred to as an SIP network. De Castro, Fig.14, [0139]; Rayo Specification, 3.

42. Starting from basic telephone services (*e.g.*, making and receiving calls), it was common for subscribers to add more features and functionality (*e.g.*, call forwarding) by using “telecommunications application services.” Marascio, 1:31-41. These telecommunications application services may be simply known as “applications,” where a computer program or application runs on a server, thereby providing the added telephone service or feature. Examples of telephone services or features that can be provided using an application include call control (*e.g.*, call redirect or forwarding). Other services and features include instant messaging and web services (*e.g.*, weather or traffic updates provided by calling a phone number associated with a computer server). De Castro, [0003], [0079]; Marascio, 1:11-2:16.

43. Telephony applications can be provided by the subscriber’s telephone service provider, as a kind of add on feature(s) to basic telephone service. For instance, I understand and personally recall that Verizon has provided and continues to provide mobile security and spam blocking services. *See also* Zapier 2, 18 (describing an “Analytic Call Tracking” application service). Applications can also be provided by various other third parties, such as software developers. De

Castro, [0113]. The iPhone App Store and/or the Google Play store, as familiar examples, have since 2008 allowed different software developers and companies to make their applications available to cellular subscribers; I understand and personally recall that some of the applications in the iPhone App Store and/or Google Play store provide telephony services such as call control. *See also* Zapier 2, 1-4 (listing example applications available to users for telephony services such as Google Talk).

44. Providing subscribers with multiple telecommunication applications services from a variety of developers was commonplace circa 2013. De Castro, [0106]. Typically, a corporate “portal” provided at a web site would be used to provide phone service subscribers with a variety of service applications and service level agreements. Robb, 34:26-31. Additionally, service providers may select applications services to include in their own portal from a product catalog. *See* Robb, 7:53-55.

B. Application Programming Interfaces (APIs)

45. It was common practice to use a computer to control and manage a phone or a telephone system. Taylor, 2:45-47. Developers and coders would create telecommunication service applications in a variety of computer programming languages. For example, Ruby, Python, Groovy, and PHP were commonly used. De Castro, [0035]. In addition, a telephony-specific API, known as TAPI, was

introduced in 1993. TAPI allowed computer applications to interact with the telephone system to, for example, place and route calls. *See e.g.*, [Newton] at pp. 673-678, 183, [Microsoft] at p. 461. For example, Microsoft Outlook used TAPI to dial a user's phone by clicking on a contact in the Microsoft Outlook user's contact list. The NBX100 telephone system that I worked on had released a system with TAPI integration prior to the time I left NBX in 1999. In order to allow these applications or software programs to communicate with other software, an interface is needed. An application programming interface (API) is such an interface that allows two pieces of software to communicate with each other. De Castro, [0022]-[0036].

46. It was common to use APIs to interface between programming languages, computer operating systems, software libraries¹, and communications

¹ A software library is a collection of resources used by a computer program. A software library may include pre-written code and routines for performing certain software functions. For instance, "cmath" is a library in C++ that includes code for performing various math functions, such as calculating a maximum value. These resources in a software library can be used by different programs. *See e.g.*, <https://cplusplus.com/reference/cmath/>; Taylor, 9:41-49; Marascio, 7:14-19, 29:13-24; de Castro, [0023], [0041], [0067], [0119].

protocols. De Castro, [0022]-[0036]; Marascio, 5:37-58; Taylor, 2:64-3:6.

Computer applications coded in different languages use different APIs to interface with, for instance, a given computer operating system; for instance, the API interfacing Ruby with Apple's iOS is different than the API interfacing Python with iOS. As another example, Ruby and Python would use different APIs to interface with a given communication protocol (*e.g.*, SIP or XMPP). *See* de Castro, [0022]-[0036]; Marascio, 5:37-58; Taylor, 2:64-3:6.

47. All of the features and functionality described in this section were well known in the early 2010s, and a POSITA would have been familiar with them.

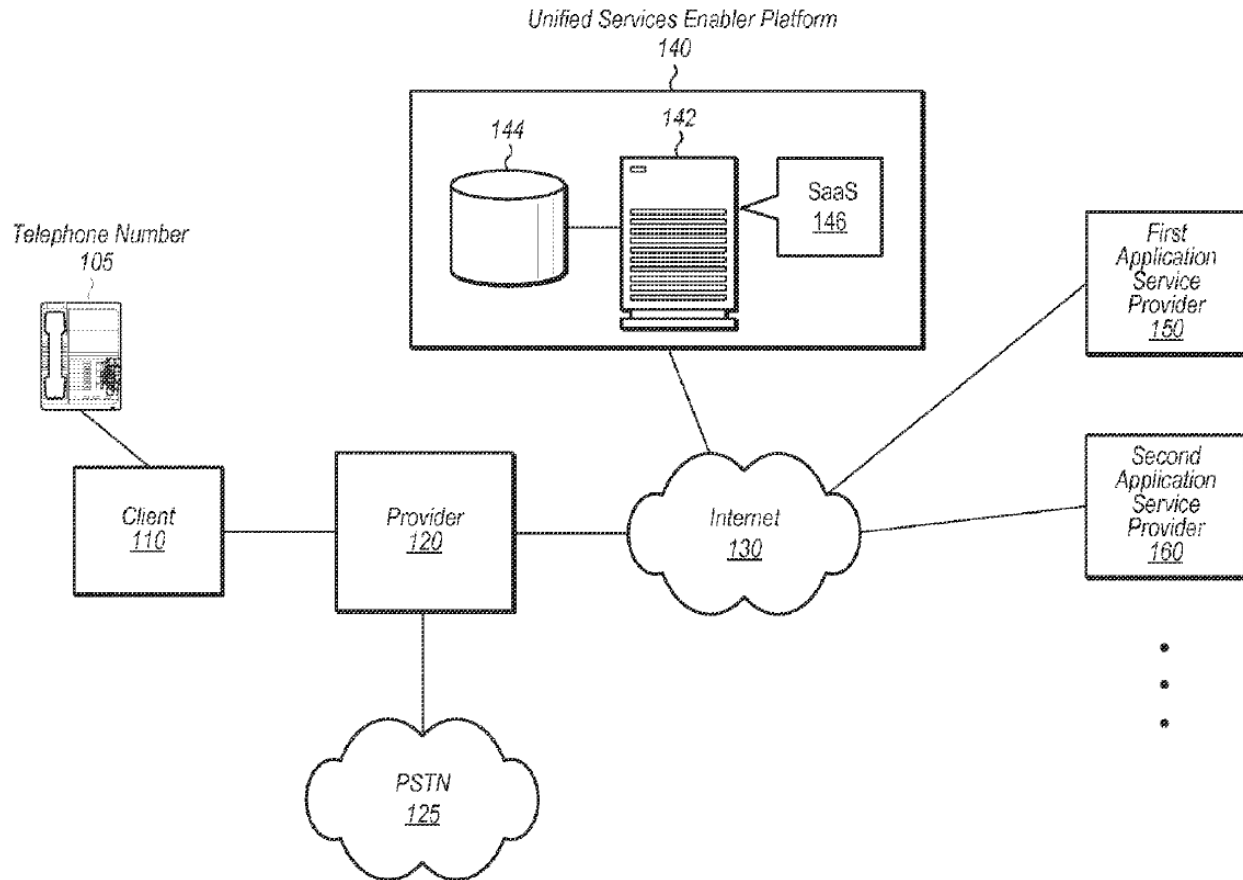
VI. OVERVIEW OF THE '048 PATENT

48. The '048 patent regards "integrating [a] plurality of API interfaces" to "provide services for use in association with a telephone number of a subscriber." '048 patent, 1:52-58. The telecommunications services are provisioned to user or subscribers from service providers with different ecosystems using a unified software platform. '048 patent, Abstract, 3:55-58, 11:37-42. The '048 patent describes examples of third-party telecommunications services and applications, including session initiation protocol (SIP) for controlling phone call sessions over internet networks, and applications for messaging, collaboration, and web services. '048 patent, 9:38-49.

49. The '048 patent then alleges: “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. '048 patent, 1:32-33, 58:45-48. The '048 patent also alleges that “new techniques for integrating advanced services and applications with legacy telephone service are desired.” '048 patent, 1:39-41. The '048 patent then purports to describe “new techniques for integrating advanced services and applications with legacy telephone service.” '048 patent, 1:34-36.

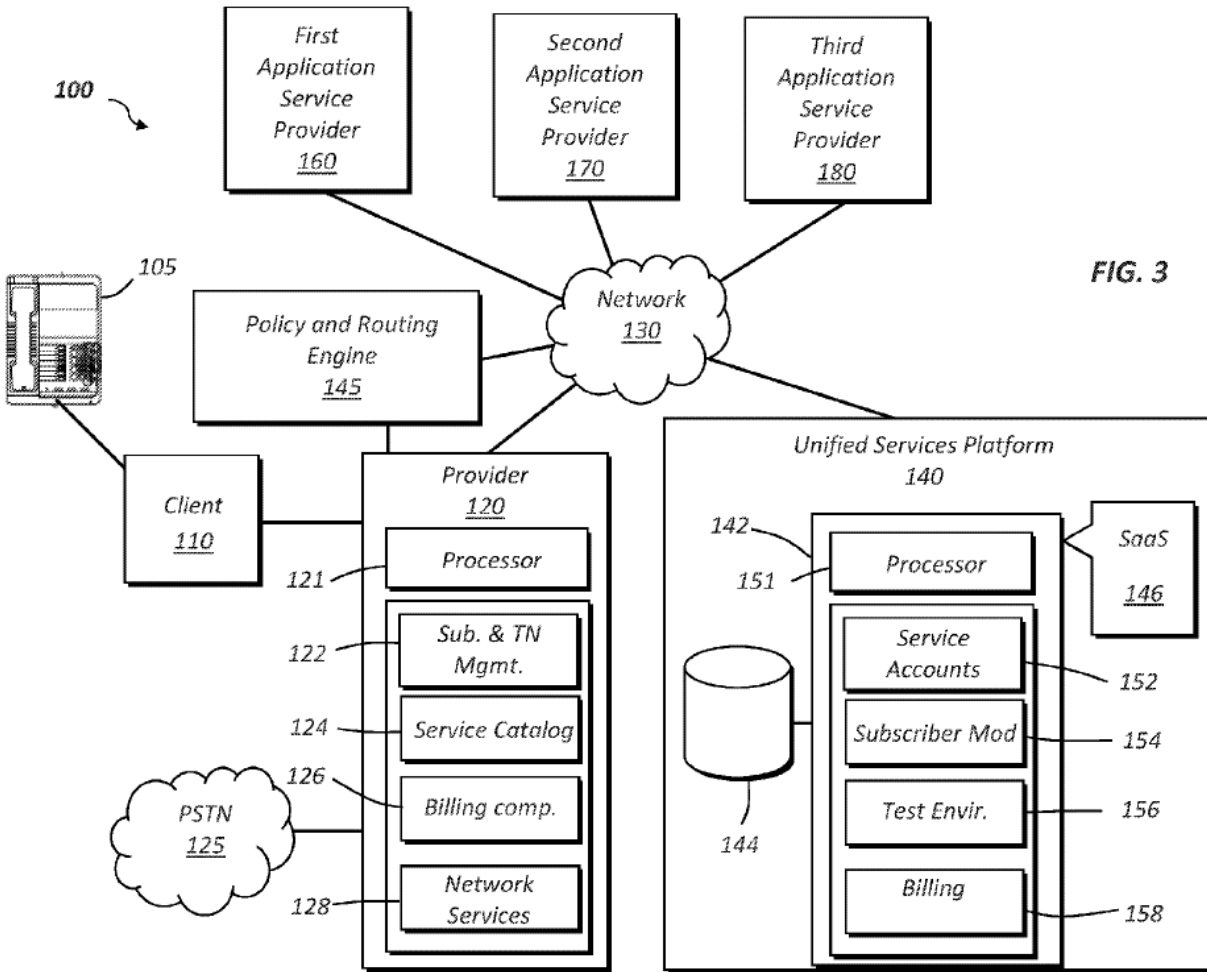
50. The '048 patent then describes “integrating [a] plurality of API interfaces” to “provide services for use in association with a telephone number of a subscriber.” '048 patent, 1:52-58. For instance, “two of the API interfaces are different, and at least one of the plurality of API interfaces from the plurality of application service provider systems is not compatible with the [network] provider API interface.” '048 patent, Abstract, 1:59-63. To address differences between API interfaces from different application service providers, the '048 patent merely describes “integrating [] each of the plurality of API interfaces with a unified services platform API interface” to “provide a common interface between the application service providers.” '048 patent, Abstract, 13:35-42.

51. Figure 1 below shows the unified services enabler platform 140 connected via internet 130 to client users 110, telephone service provider 120, and application service providers 150, 160. '048 patent, Fig.1, 3:35-37, 4:14-28.



'048 patent, Fig.1.

52. Figure 3 below illustrates unified services platform 140 providing a common or unified API and connected via network 130 to client users 110, telephone service provider 120, and application service providers 160 and 170. '048 patent, Fig.3, 10:26-31, 10:51-52, 13:35-38.



'048 Patent, Fig.3

53. However, all of these concepts were well known to a POSITA as of the priority date of the '048 patent. Specifically, it was well-known and described at length in the prior art to integrate services from different ecosystems and having different APIs for a single subscriber/user. It was also well-known in the prior art to associate a user's telephone number with different services. De Castro, [0138]-[0141].

VII. CLAIM CONSTRUCTION

54. It is my understanding that in order to properly evaluate the '048 patent, the terms of the claims must first be interpreted. It is my understanding that for the purposes of this *inter partes* review, the claims are to be construed under the so-called *Phillips* standard, under which claim terms are given their ordinary and customary meaning as would have been understood by a POSITA in light of the specification and prosecution history, unless the inventor has set forth a special meaning for a term. I have also been informed that claim terms only need to be construed to the extent necessary to resolve the obviousness inquiry. I have reviewed the entirety of the '048 patent, as well as its prosecution history.

55. It is my understanding that no terms in the Challenged Claims require specific construction. While express constructions are not necessary for the purposes of this proceeding, the '048 specification describes examples related to the claim terms “*definition of a service*” and “*definition of [] a plan*” that are relevant to the prior art in this proceeding:

The services can be defined through the use of catalog data such as a **service name, [] description, [] availability**, and the like. ... The plans comprise information regarding pricing, features, and/or **service limits associated with the service** offered by the application service provider

160.

'048 patent, 15:65-16:8. Here, the '048 patent describes a service name, service description, and service availability as examples of “*a definition of a service*”; further, “service limits associated with the service” is an example of “*a definition of [] a plan.*” '048 patent, 15:65-16:8.

VIII. IDENTIFICATION OF HOW THE CLAIMS ARE UNPATENABLE

56. The discussion in this Declaration provides a detailed analysis of how the asserted prior art references teach each limitation of the Challenged Claims.

57. As part of my analysis, I have considered, and discuss in detail, the scope and content of the prior art and any differences between the alleged invention and the prior art.

58. It is my opinion that the alleged invention recited in the Challenged Claims would have been obvious in view of the teachings of the asserted prior art and the knowledge of a POSITA.

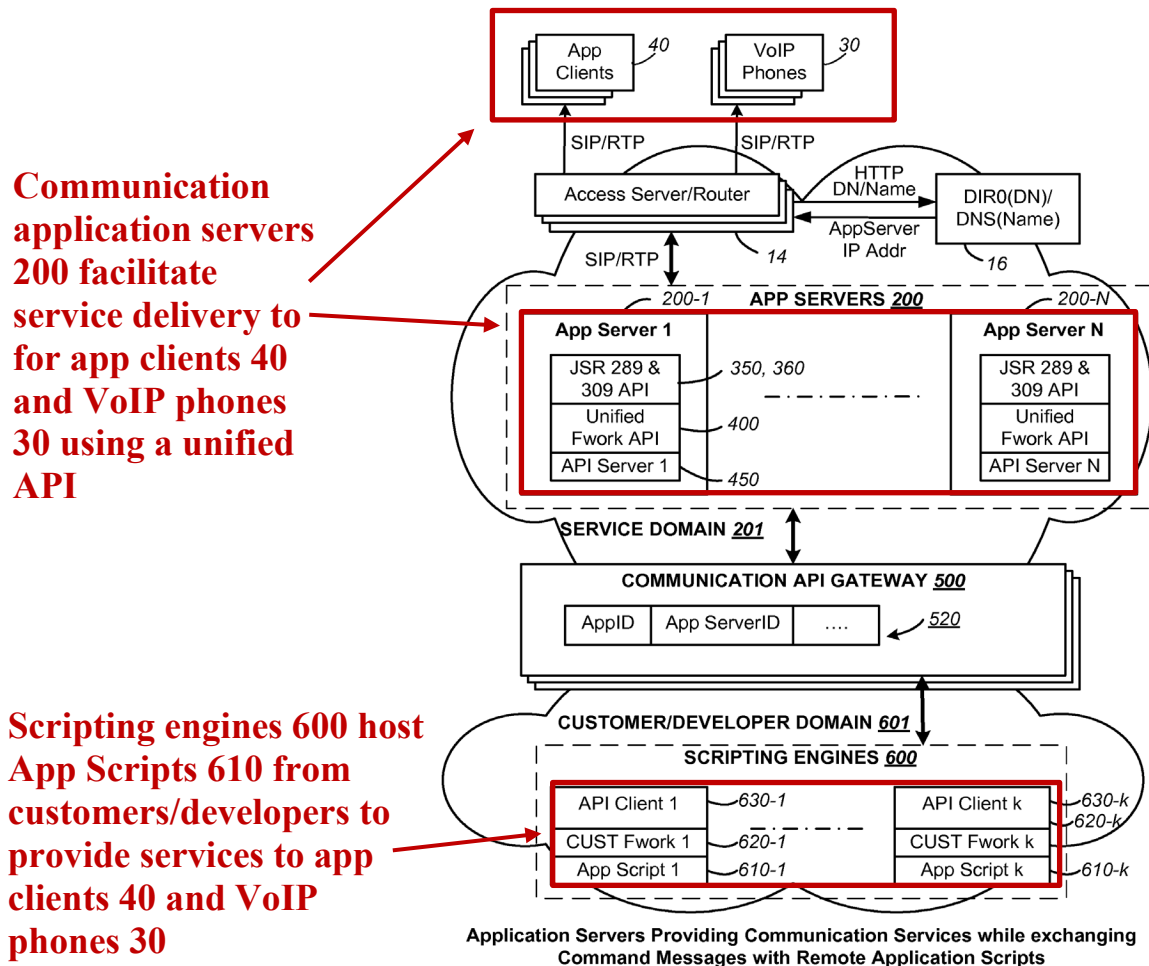
IX. THE COMBINATION OF DE CASTRO AND ROBB RENDERS OBVIOUS CLAIMS 13, 14, AND 17

A. Summary of de Castro

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

60. De Castro's communication application servers communicate with "a plurality of scripting engines 600." De Castro, [0119]. These scripting engines are used to "host and process the customers' applications scripts 610." De Castro, [0119]. Here, the term "customer" does not refer to the service user but instead to a "developer" that provides a service by coding the application script. De Castro, [0039], [0041], [0071], [0113]-[0114].

61. De Castro illustrates, in Fig.14 shown below, VoIP phones 30 and application clients 40 (i.e., subscribers or users) connected to scripting engines 600 via a SIP/RTP network and communication application servers 200. De Castro, Fig.14.



De Castro, Fig.14 (annotated)

62. The communication application servers 200 receive commands from the application scripts and scripting engines. De Castro, [0038]. In particular, the communication application servers use “a messaging API protocol [that] enables an application script to pass commands remotely to the communication application server to operate it.” De Castro, [0038]. With this messaging API protocol or “unified API,” 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.” De Castro, [0038], [0041].

63. An example messaging API protocol is the “Rayo protocol” which “primarily deals with calls” and telecommunications services associated with a user’s phone number. De Castro, [0138]-[0141]. De Castro teaches that API server 450 executes an application via the Rayo protocol to control a call using a JID:

The following are examples of **the commands** supported by the unified framework API 400 through the API server 450 and a specification of the messaging API protocol (hereinafter referred to as the “Rayo™” protocol).

Calls

The Rayo protocol primarily deals with calls. Inbound calls originate from the PSTN or via SIP and are offered to Rayo clients² via XMPP using a Jabber Identifier (JID). **Each call is in turn represented by its**

² In Fig.14 and the corresponding disclosure (e.g., [0115], [0116]), de Castro refers to the end users of the App scripts as “clients,” such as “App Clients 40.” These “App Clients 40” should not be confused with the “Rayo clients” described in [0141] of de Castro, for example. The “Rayo clients” correspond to the customers/developers whose app scripts are hosted by the App servers 200. The “Rayo server” corresponds to an App server 200.

own unique JID allowing a two way conversation between the Rayo client and the server that's handling the call signaling and media.

De Castro, [0138]-[0139].

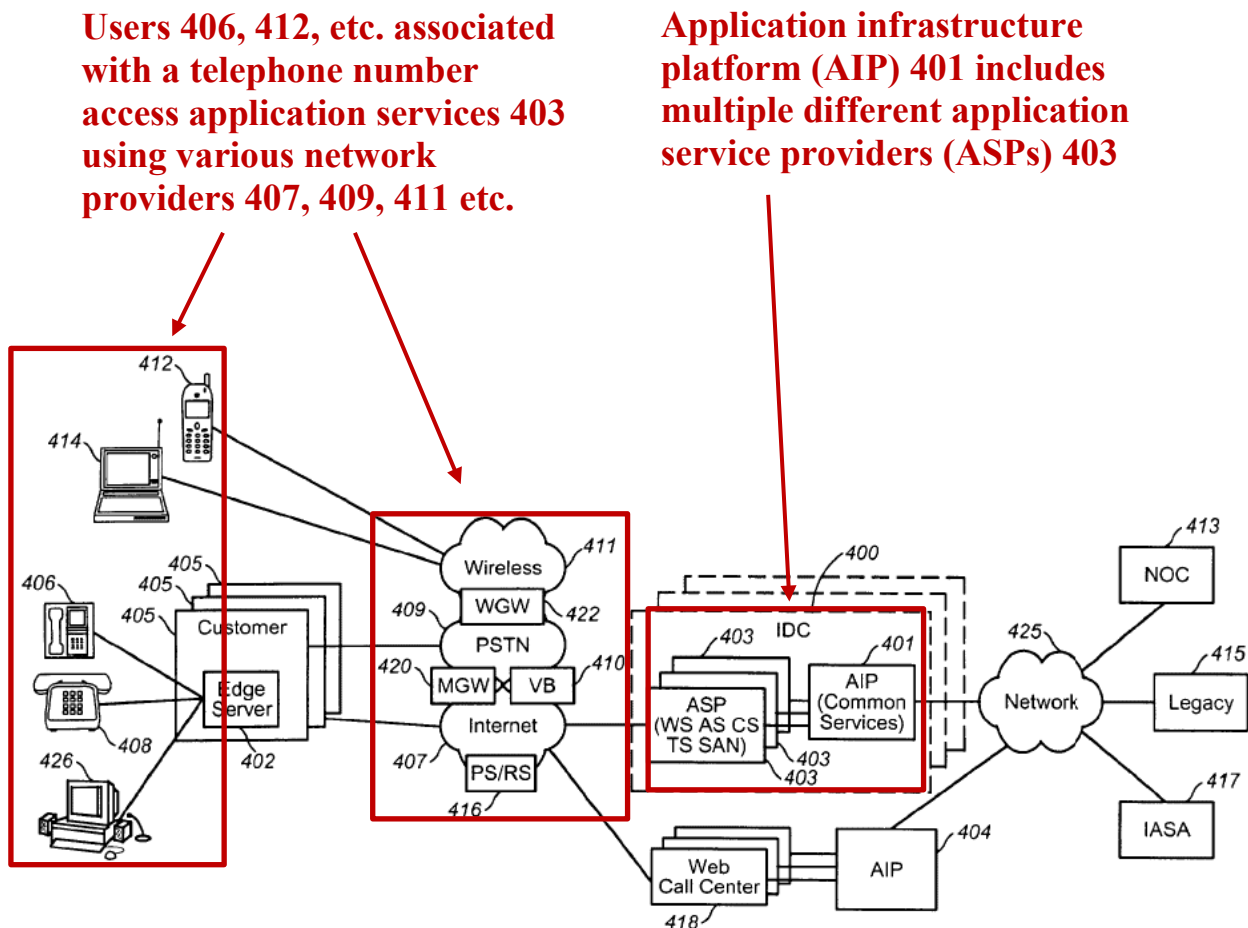
64. De Castro is related art to the '048 patent and is from the same technological field: providing communications services. De Castro, [0002]; '048 patent, 1:23-26. For instance, de Castro describes a “unified framework” used to provide application clients access to a variety of communication applications. De Castro, Abstract. Likewise, the '048 patent provides a “unified services platform” to access “a variety of services and applications” from “multiple different ecosystems.” '048 patent, 13:60-63, 9:60-62, 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* '048 patent, 4:60-62, Abstract; de Castro, Abstract. De Castro is also related to the problem described in the '048 patent. For example, de Castro describes how service providers may use their own APIs to provide communications services to clients by telephone. *See* '048 patent, Abstract, 11:33-42, 2:1-5, 4:13-17; de Castro, Abstract, [0139]-[0142].

B. Summary of Robb

65. Robb describes techniques for providing “services and data to end user devices,” such application services to subscribers using telephones and personal digital assistants. Robb, 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., users) who are associated with telephone numbers. Robb, Abstract, Fig.5.

66. Robb illustrates, in Fig.5 shown below, telephone users (e.g., 406 and 412) accessing AIP 401. Robb, Fig. 5, 15:56-16:9. Here, AIP 401 includes multiple different ASPs 403 via different network providers (e.g., Internet 407, and public switched telephone network 409). Robb, Fig. 5, 15:56-16:9, 16:31-33.



Robb, Fig.5 (annotated)

67. Robb further describes techniques for “display[ing] the services that the user can access via Corporate Portal 3109.” Robb, 35:9-10. Robb teaches that “the AIP 114 provides a collection of applications and services to the hosted ASPs 102.” Robb, 7:29-30. Further, “[t]he AIP 114 combines (i.e., integrates) applications, services, content to ASPs 124 ... These applications and services are customized by the ASP 124 for rent or resale to customers of the ASP 124.” The AIP facilitates the selection of customization of wide variety of services for resale by “aggregat[ing] its own applications and services with those of both external ASPs/Internet Service Vendors (ISVs) and its own ASPs into a ‘Product Catalog’.” Robb, 7:53-55. “[T]he ASPs 102 in turn may repackage and resell to their customers as corporate portals.” Robb, 7:30-32.

68. 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.” Robb, 26:47-49. To allow ASPs to leverage the variety of services available, the AIP’s catalog and customization features facilitate the “setting up of service packages” by the ASPs for resale. Service packages, which Robb also refers to as “service packs,” are “an aggregation of services for a user role” and include “SLA [service-level agreement] terms and conditions to the service.” Robb, 31:40-41, 34:16-17, 34:25-27.

69. Robb also describes testing applications and services, including ensuring that applications and services are properly integrated. This is accomplished via a “Staging and Test Bed environment 503.” *See* Robb, 17:2-9. 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.” Robb, 17:5-9. The testing process described in Robb includes testing events and corresponding work flows implemented via 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.” Robb, 10:53-55. After testing each new application, the new application is deployed allowing access to the new application. *See* Robb, 7:38-41, 17:5-9.

70. Robb is related art to the '048 patent and is from the same technological field: providing communications services. Robb, 3:40-42; '048 patent, 1:23-26. For instance, Robb describes a “application infrastructure platform” that is used to provide application clients access to “multiple...application services.” Robb, Abstract. Likewise, the '048 patent provides a “unified services platform” for “a variety of services and applications” from “multiple different ecosystems.” '048 patent, Abstract, 13:60-63, 9:60-62. 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*

'048 patent, 4:60-62, Abstract; Robb, 15:56-16:9. Robb is also related to the problem described by 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* '048 patent, Abstract, 11:33-42; Robb, 6:24-35. Robb also describes how service providers may publish services on a portal for telephone clients. *See* '048 patent, 12:9-16; Robb, 34:16-17.

C. REASONS TO COMBINE DE CASTRO WITH ROBB

71. A POSITA considering the de Castro reference would have also looked to Robb because they are related art in the same technological field. *See e.g.*, De Castro, [0039]; Robb, Fig.5. For instance, these references relate to integrating multiple telecommunications application services and network providers for end users associated with a telephone number. *See e.g.*, de Castro, Fig.14, [0139]-[0141]; Robb, Figs.2, 5, 9:43-50.

72. It would have been obvious, beneficial, and predictable to use Robb's teachings regarding aggregating telecommunication services in a product catalog and repackaging and reselling the cataloged services through a corporate portal in combination with de Castro's system. For example, a POSITA would have found it beneficial to publish Rayo protocol services from other customer/developers on a product catalog accessible by other customer/developers for repackaging,

customization, and reselling to de Castro's users/subscribers via a corporate portal, per Robb.

73. De Castro already discloses several concepts to which Robb's teachings apply: developers coding application scripts; scripts including commands that are expressed in Rayo protocol messages; and simplifying the telecommunications application development process to allow the developer to "focus on the business logic." De Castro, [0038], [0041], [0098]. In de Castro's system, developers can "code the application script in a preferred programming language," which is "expressed in terms of the finite set of commands" of a unified API like the Rayo protocol. De Castro, [0041], [0138]-[0141]. De Castro also teaches that the unified API or Rayo protocol thereby "allows application clients 40 such as, for example, VoIP phones to access the applications hosted in the communication application servers." De Castro, [0139], [0115].

74. A POSITA would have recognized many computer engineering-related advantages to using the Rayo Protocol, including that there are "commands supported by the unified framework API 400 through the API server 450 and a specification of the messaging API protocol (hereinafter referred to as the 'Rayo™' protocol)." De Castro, [0138]. Here, "the specification of the ... Rayo protocol" refers to the The Rayo Specification (XEP-0327: Rayo) (Ex.1009), which was also drafted by the named inventor of the de Castro U.S. patent

publication. De Castro, [0138]; de Castro Declaration (Ex.1021), 2-3 (referring to Ex.1009 as the “Rayo Standard”).

75. The Rayo Specification describes the benefits of using the Rayo protocol as a unified API: “3PCC [third-party call control] activity is [] done using process-internal APIs rather than a wire protocol like XMPP.” Rayo Specification, [3], [16]. A POSITA would have recognized that using a process internal API call would be faster than using a protocol over a wire which a POSITA would understand would be externally over a network. A POSITA would have found it beneficial to use the Rayo protocol in a system for deploying telecommunication services to end users via a portal, as described by Robb. *See* Rayo Specification, [3], [16]. Likewise, a POSITA would have been motivated to use Robb’s portal technique in combination with de Castro’s system for provisioning application services using a communication application server and scripting engines. *See* de Castro, Fig.14.

76. Robb provides a technique for how a platform that integrates multiple telecommunications application services from different providers (like de Castro’s communication application server) publishes the services that are available to end users. Robb, 34:16-17, 34:25-27, 35:9-10. For instance, Robb describes aggregating services to a product catalog accessible by other service providers. The other service providers can access the product catalog to select and customize the

available services into “service packs” on a “corporate portal.” Robb, 34:16-17, 34:25-27. A POSITA would have found it obvious to make Rayo-protocol services available to de Castro’s clients via a portal website, which includes service packs that describe the services, provide application code/software, and service-level agreements. *See* de Castro, Fig.14; Robb, 34:16-17, 34:25-27.

77. Robb explains how testing may be performed to ensure the accuracy of application software as well as the integration of application software with other applications or services within an environment prior to deployment. Robb, 7:38-41, 16:59-17:30, 33:49-51, 34:29-31. To perform testing, Robb provides “a tested and integrated environment.” Robb 8:42-44. As testing has been a necessary part of any software development since its inception, a POSITA at the time of the invention would have recognized the need to include comprehensive testing of applications to systems like de Castro’s.

78. A POSITA would have recognized the benefits of the resulting combination—*i.e.*, aggregating services in a service provider-facing product catalog publishing customized service packages for use on user-facing corporate portals by de Castro’s system, as taught by Robb. *See* De Castro, Fig.14, [0041], [0138]-[0141]; Robb, Fig.5, 34:16-17, 34:25-27. In this regard, 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.” Robb, 26:48-49. 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. Robb, 9:66-67. This advantageously expands the range of telecommunication services and service packages available to both service providers and end users, and also facilitates service discovery by users. Robb, 6:28-32.

79. A POSITA also would have recognized using a corporate portal that publishes service packs in de Castro’s system is beneficial because the “services offered through a corporate portal have also grown more robust.” Robb, 6:53-55. Robb, for instance, further 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.” Robb, 6:53-57. Robb’s technique thereby beneficially provides de Castro’s system a flexible and user-friendly implementation. That is 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.” Robb, 10:1-5. In offering telephony services directly to an end user 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.

80. The combination of de Castro and Robb is simply the ordinary use of common service cataloging techniques described by Robb as well as other prior art references (*e.g.*, Zapier 1 and Zapier 2), as applied to a similar unified-platform system (*e.g.*, de Castro's unified API). Indeed, this combination would yield predictable results with an expectation of success (*e.g.*, publishing Rayo-protocol application services on a product catalog and a portal so they are accessible to other service providers and end users). *See generally* de Castro and Robb. Robb's portal technique was also specifically intended to be implemented in servers integrating multiple telecommunications applications like de Castro's. *See* De Castro, Fig.14, [0038], [0041]; Robb, Fig.5, 34:16-17, 34:25-27. Each and all of these reasons would have motivated a POSITA to combine the references to arrive at the invention recited in the Challenged Claims.

81. 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. That is, both references describe providing multiple application services on a unified platform, as well as telephone users accessing different application services over telecommunications and internet

networks such as an SIP network. *See* De Castro, Fig.14; Robb, Fig.5, 9:49-50.

Further, Robb explains that, “[w]ith respect to Infrastructure, the following systems are supported: ... API/Interfaces ... and SIP Integration.” Robb, 20:33-55. Thus, Robb describes that its teachings apply in the context of integrating several API/interfaces as well as SIP support in providing multiple services to subscribers. Robb, 20:33-55. This teaching from Robb parallels de Castro’s teachings about providing a unified API that integrates APIs of multiple third-party applications as well as integrates an SIP API. *Id.*; *see also* de Castro, Fig.14. And like de Castro, Robb describes “integrating a service specific to one of the plurality of the providers, and selectively presenting the specific service.” Robb, 3:31-39.

82. A POSITA also would have found this combination predictable and obvious because publishing available telecommunication application services so that they are accessible to end users was long known in the art. *See e.g.*, Zapier 1, 1 (“The Service Summary page is the screen you are presented with immediately after creating a new service. Fields include Title, Description, and Image. Title should be the short human-readable version of your service. ... Description is a short few-sentence description of what your service primar[ily] does.”); Zapier 2, 1 (illustrating services such as AIM, AlphaMail, etc., published on a portal and organizable by service name or category).

83. Modification(s) to de Castro's teachings, including implementing software programming or hardware, to accommodate the teachings of Robb, if necessary, would have been within the level of ordinary skill in the art. That is, a POSITA would have had computer programming and web design skills for aggregating application services (e.g. Rayo protocol services) into a product catalog, and publishing selected services from the catalog on a corporate portal. Using the implementation details and techniques provided by Robb, in combination with de Castro's unified platform, might involve simple computer programming changes, which would be well-known to a POSITA. The computer programming changes would involve aggregating the Rayo-compatible services into a product catalog and publishing selected services and combinations of services on a portal as described by Robb. This software change would have been predictable given that both Robb and de Castro describe providing telecommunications services from different providers to a subscriber. *See de Castro*, [0139]-[0142]; *Robb*, 6:28-32; *see also Zapier 1 and Zapier 2* (quoted above).

D. Detailed Analysis of Claims

1. Claim 13

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

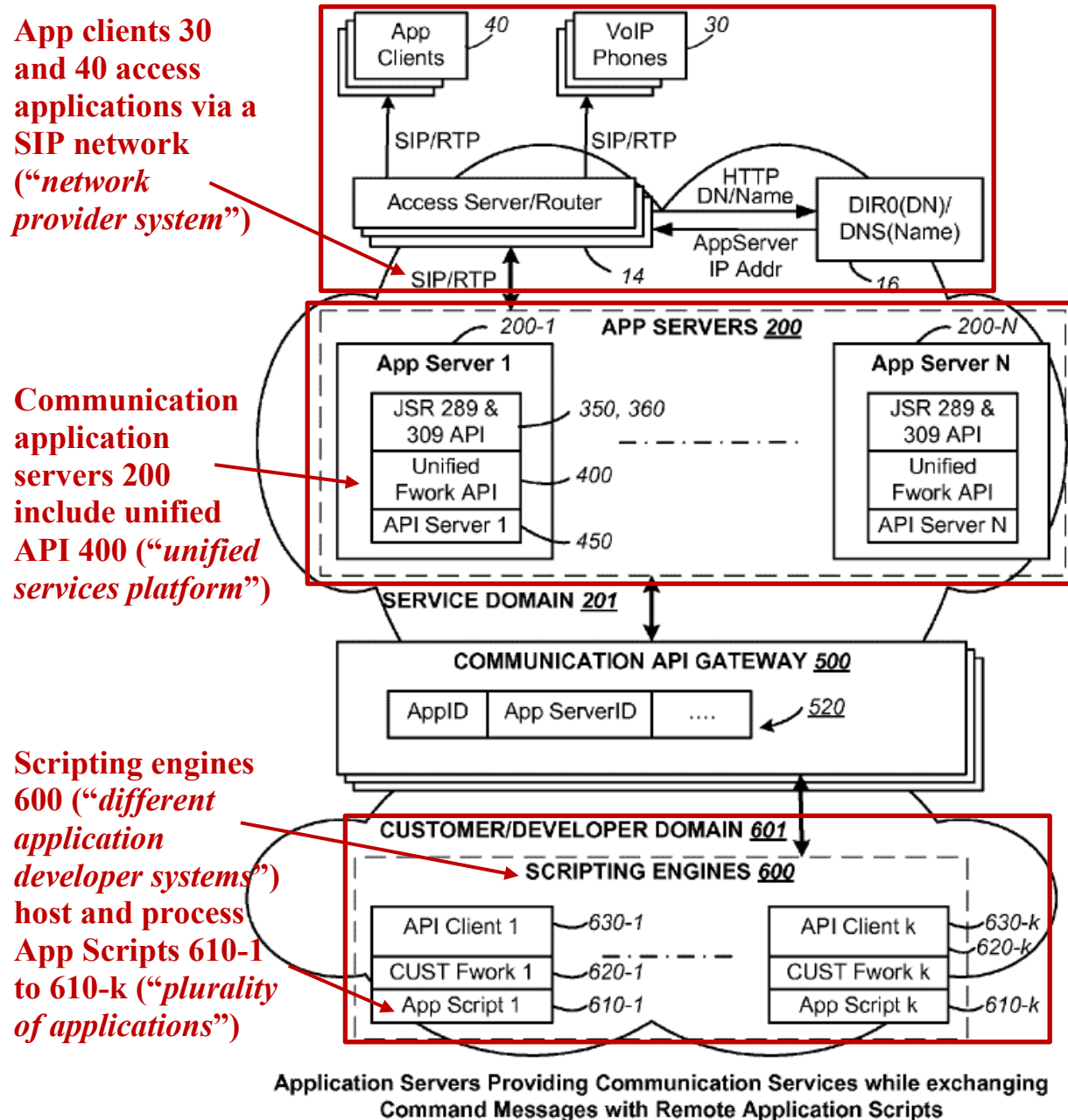
84. De Castro's system includes a "**unified framework**" ("*unified services platform*") that "supports a **unified API**." De Castro, [0038]-[0039]. The unified framework facilitates the deployment and execution of "**application scripts**" providing "telephony and communication services" for clients over a network ("*provisioning a plurality of applications*"). De Castro, Abstract, [0039], [0115], [0119].

85. These application scripts are provided by a plurality of "developers" who are also referred to as "customers with application scripts." De Castro, [0035], [0039]. "[A]pplication scripts [are] processed remotely by **appropriate scripting engines**." De Castro, [0038]. These "**third party scripting engines**" are associated with the plurality of third-party developers ("*different application developer systems*"). De Castro, [0119], [0041].

86. In de Castro's system, clients access the telecommunications application services (i.e., services for controlling phone calls) hosted in the communication application servers "**via SIP**" networks ("*network provider system*"). De Castro, [0139]. De Castro's system thus provides a unified framework that "allows application clients ... to access the application hosted in the communication application servers" De Castro, [0115]. In particular, the communication application servers communicate with the scripting engines using a "messaging API protocol" that expresses commands from the scripting engines

that are compatible with and processed by the “unified framework API” at the communication application servers. De Castro, [0119].

87. Fig.14 illustrates App Scripts 610-1 to 610-k (“*applications*”) provided to SIP-network clients 40 and 30 via communication application servers 200. De Castro, Fig.14. It also illustrates that communication application servers 200 include unified API 400 (“*unified services platform*”). De Castro, Fig.14.



De Castro, Fig.14 (annotated)

88. Robb also details the process of creating and testing a new application service with a unified service platform. Specifically, Robb outlines an “application infrastructure platform (AIP)” that is used to allow application clients to access “multiple application services.” Robb, Abstract. This platform includes various

“application service provider (ASP) systems” that provide services to customers associated with telephone numbers. Robb, Abstract, Fig. 5.

The AIP system and methodology combines a development and execution environment into one set of capabilities to support both aspects of the operations of its customers, such as ASPs. The development environment provides the **tools, component libraries, and Software Development Kits (SDKs)** needed to **rapidly build applications** to leverage common services.

Robb, 12: 33-39.

89. Robb also describes testing the applications created. In particular, Robb illustrates a Staging/Test Bed 503 in Fig. 6A, reproduced below.

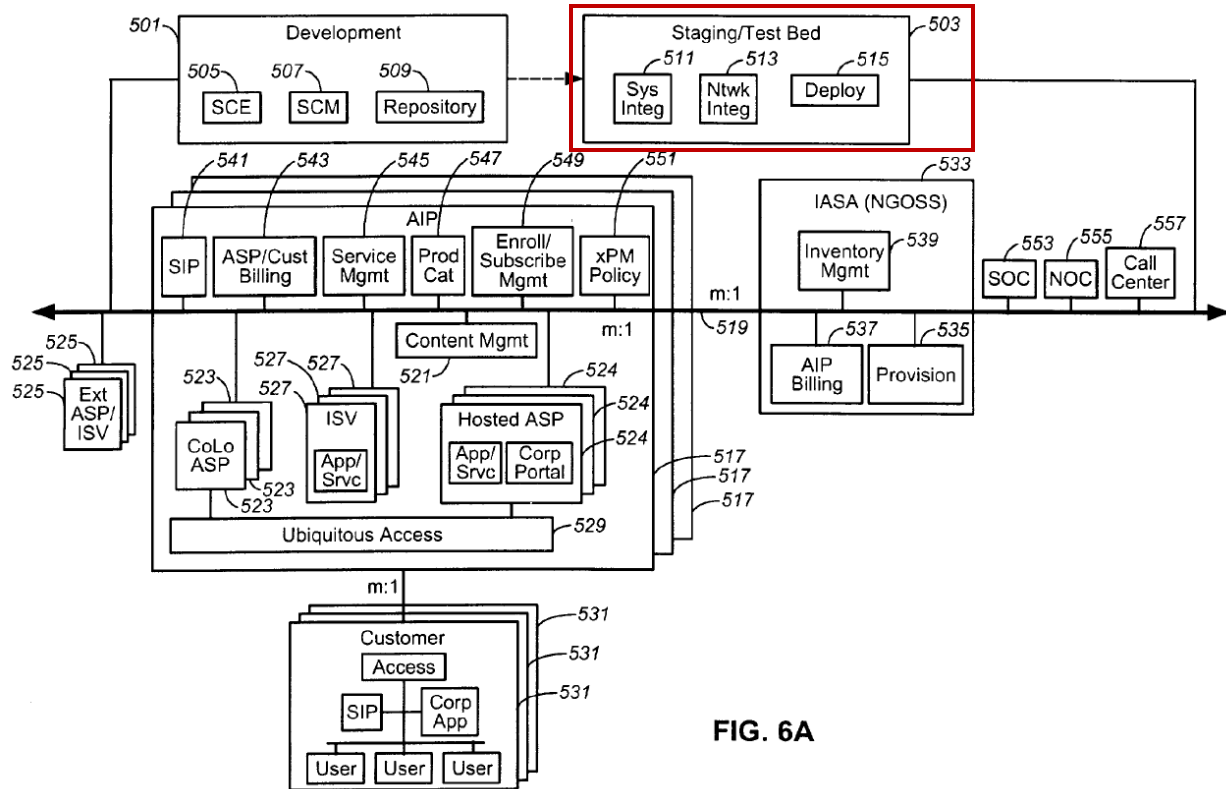


FIG. 6A

Robb, Fig. 6A (annotated)

The Staging/Test Bed 503 is positioned within Fig. 6A after the Development environment 501 and is used to test new applications prior to deployment.

The Staging and Test Bed system 503 includes System Integration or System Integrator (SI) 511, Network Integration 513, and Deployment 515, all of which may be located at different sites. 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.

Robb, 17:2-9. After testing, “[a]n external one or more ASP/ISVs 525 is connected to the message bus 519, and may serve to make development,

staging and testbed produced applications, and software available to the AIP for inclusion in its available services.” Robb, 17:26-30; see also Robb, Abstract (“The AIP is associated with a development and test bed environment which conducts product and service development and testing.”).

90. Thus, de Castro teaches customer/developer applications, which are processed by scripting engines, providing services to clients on a SIP network via communication application servers via a unified API. In addition, Robb describes an application infrastructure platform in which “integration and tests are executed.” Robb, 31:22-23. The combination of de Castro and Robb discloses a method “*comprising*” the recited limitations as detailed below.

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

91. In the system of de Castro, there are a plurality of “**scripting engines**” for “application scripts to be processed remotely.” De Castro, [0038], Fig.14.

92. These scripting engines (“*plurality of application service provider systems*”) provide application services. De Castro, [0130]. Each application is identified by an application identifier value or “**AppID**.” De Castro, Fig.14, Fig.19. Information about these services is registered in a “**registry**.” De Castro, Fig.14, Fig.19, [0130]. That is, the registry stores information about each application, organized by the applications’ AppIDs. De Castro, Fig.14, Fig.19, [0130].

93. De Castro describes this registration process in more detail; for instance, “**registry 520...registers...the customers’ applications in the customer/developer domain.**” De Castro, [0130]. The registry also logs data such as “**the properties and status associated with each API clients 630.**” De Castro, [0130]. That is, 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.” De Castro, [0119], [0127].

94. In Fig.19 shown below, de Castro illustrates registry 520 logging information including the “properties/status” about each scripting engine’s API client 630 identified by an “App ID.” De Castro, 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



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

De Castro, Fig.19 (annotated)

95. A POSITA would have understood that the information stored in the registry for each service is a “*service account*[].”

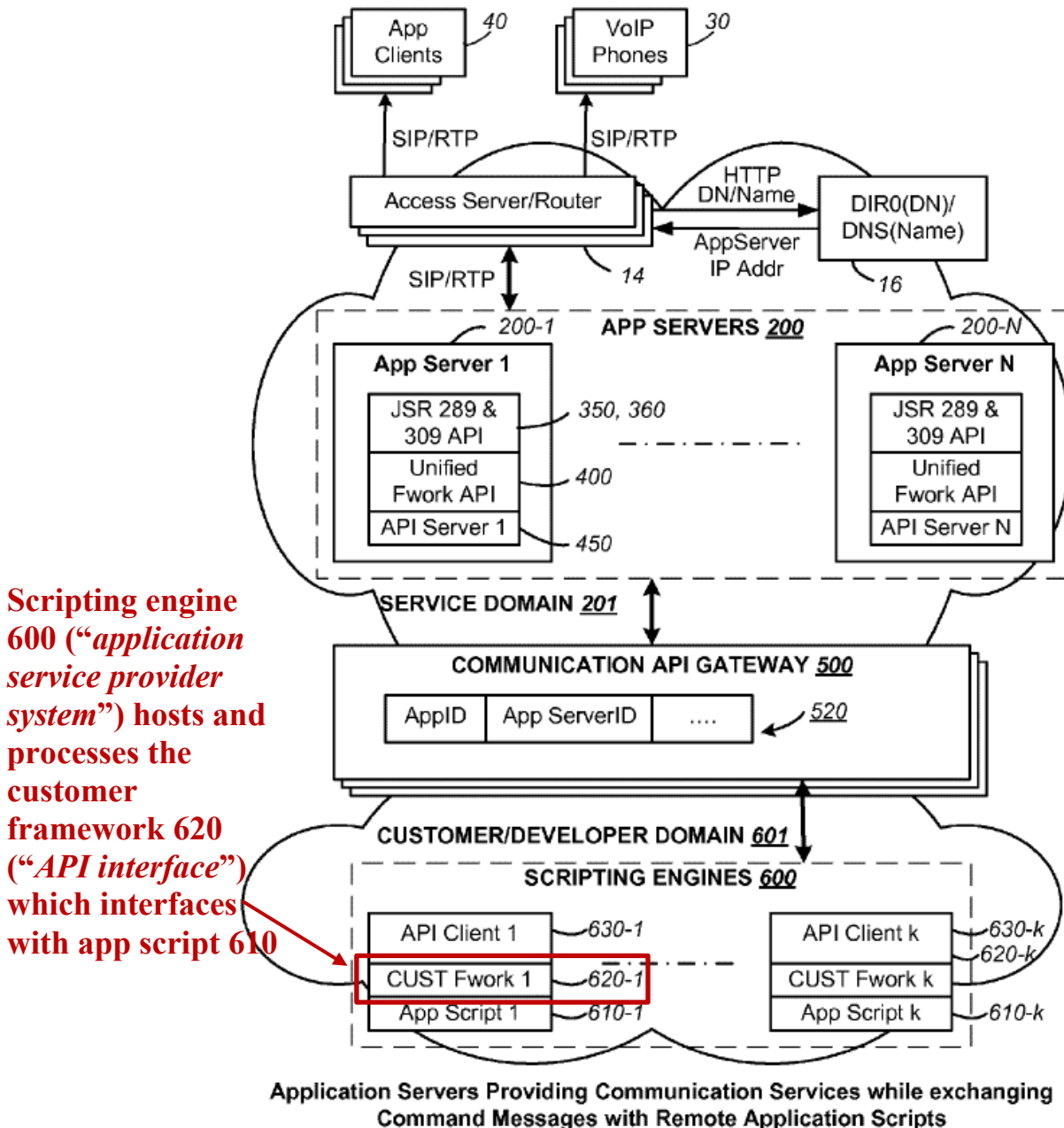
96. Thus, de Castro teaches a registry that registers and logs information (“*service accounts*”) about the scripting engine of each application service (“*application service provider systems*”), including logging application properties and status information, which renders obvious “*establishing a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

[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,

97. In de Castro, each scripting engine (“*application service provider system*”) includes a “**custom framework**” to interface with an application script. De Castro, [0041], [0119], [0067].

98. De Castro explains that a developer, also referred to as a customer, “can code the application script in a preferred programming language using a **custom framework providing a set of customer-specific libraries.**” De Castro, [0041]. Each of these custom “frameworks” used by a developer is an example of an “API” (“*a plurality of API interfaces*”). De Castro, [0067]. For instance, de Castro explains that “[f]**rameworks** are a special case of software libraries in that they are reusable abstractions of code wrapped in a **well-defined API.**” De Castro, [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.” De Castro, [0119].

99. 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. De Castro, Fig.14, [0113], [0119].



De Castro, Fig.14 (annotated)

100. De Castro also describes the application scripts and customer framework APIs as being “hosted” by the scripting engines. De Castro, [0039], [0110], [0115], [0119]. For instance, Fig.14 above shows customer framework APIs (CUST Fwork 620-1 to 620-k) hosted in scripting engines (600-1 to 600-k).

101. Because the customer framework APIs are located within and hosted by the scripting engines, a POSITA would have understood that the customer framework APIs must have been “*receiv[ed]*” by the scripting engines at some prior point in time. In other words, the scripting engines include customer frameworks that implement customer-specific APIs, and the scripting engines must have “*receiv[ed]*” the customer frameworks in order for the scripting engines to be able to include and host these frameworks and thus customer-specific APIs. De Castro, Fig.14 (illustrating each scripting engine 600 as including Cust Fwork 620).

102. De Castro also explains that application scripts and customer framework APIs are “**processed remotely** by appropriate scripting engines.” De Castro, Fig.14, [0038]. A POSITA would have understood this disclosure to mean that the scripting engines use at least one computer processor (“*a processor system*”). 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.” De Castro, [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.” De Castro, [0119]. Therefore, the scripting engines’ hosting and processing of customer framework APIs renders obvious “*receiving, by a processor system, a plurality of API interfaces.*” See De Castro, Fig.14, [0038], [0119].

103. De Castro also teaches that the customer framework APIs are different (“*at least two of the API interfaces are different*”). De Castro, [0039]. Each of the developers can code the application script in a “**preferred programming language using a custom framework providing a set of customer-specific libraries.**” De Castro, [0041]. De Castro includes as example programming languages “Ruby, Python, Groovy and PHP,” which are just some that were well known in the art. De Castro, [0035]. Other well-known programming languages include Perl and C++. 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. De Castro, [0067], [0041].

104. It was common practice in the art and well known to a POSITA for telecommunications application services provided by third parties to use “[*different*] *API interfaces*.” For instance, Taylor (Ex.1011) explains it was known to use a “set of APIs” as “procedures, protocols and tools for building software applications. These APIs are generally proprietary and specific to the individual hardware manufacturers. Developing an application [involves] ... coding the application in C++ or JAVA programming language and employing and invoking the APIs specific to the hardware.” Taylor, 2:65-3:1. In view of these teachings, a POSITA was aware that the various applications in de Castro use “specific” APIs that are often “proprietary,” as explained by Taylor. *Id.*

105. 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 a plurality of API interfaces from the plurality of application service provider systems, wherein at least two of the API interfaces are different.*”

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

106. In de Castro’s system, each “communication application server is provided with a **unified framework**.” De Castro, [0109]. In particular, de Castro’s unified framework “**supports a unified API** having class objects and functions conforming to a telephony object model.” De Castro, [0069]. The “**unified**

framework provides a unified API that gives access to a set of unified class objects for call control and media control.” De Castro, [0100].

107. De Castro explains that “the communication application server 200 does not proceed to retrieve a script for the specific communication application,” but rather “it **exposes the unified communication framework API 400 as a finite set of commands.**” De Castro, [0118]. A POSITA would have understood that this unified API teaches the “*unified services platform API interface,*” as claimed.

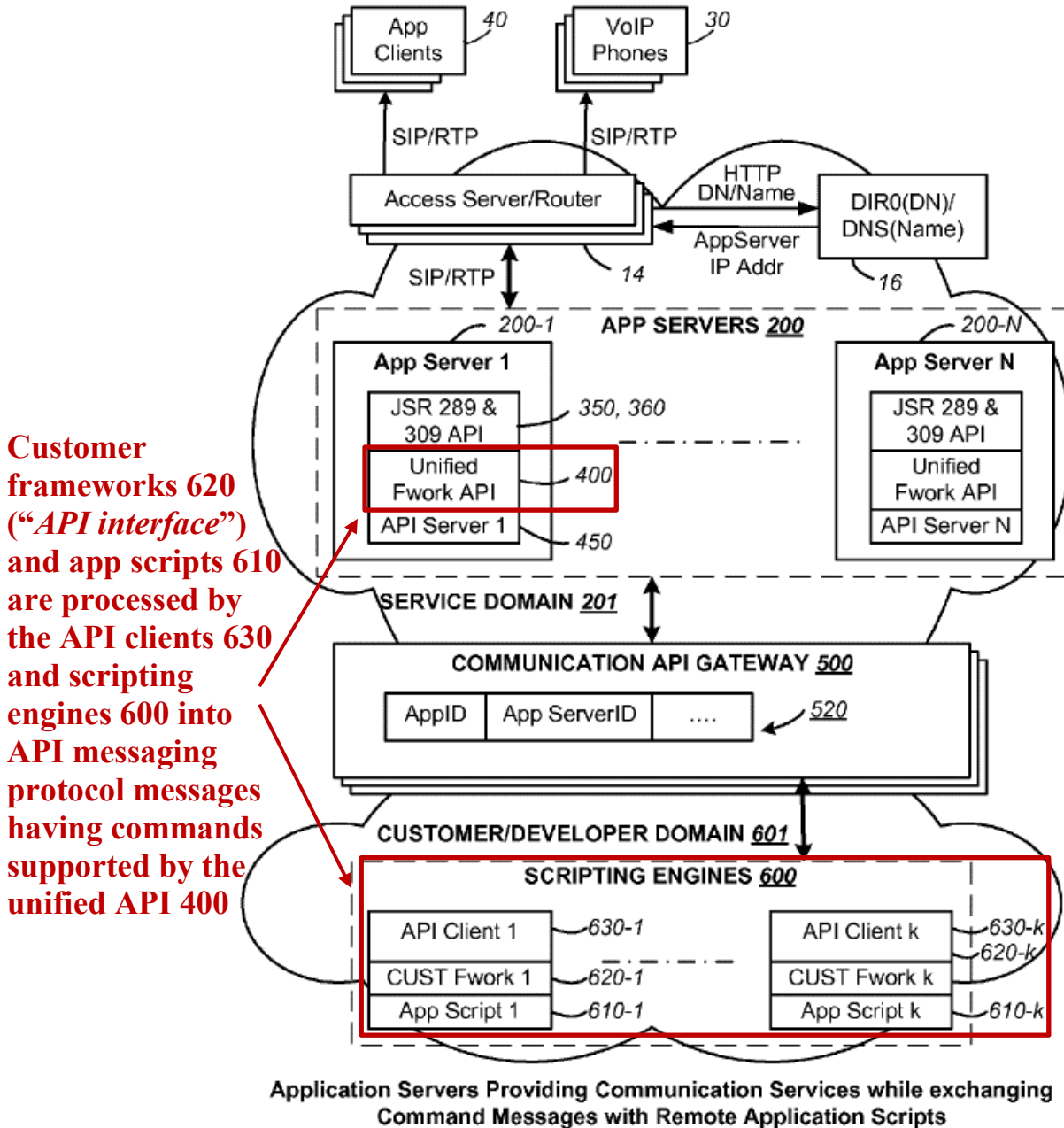
108. Further, as discussed at [13.2], a POSITA would understand that each of de Castro’s scripting engines includes at least one computer processor (“*processor system*”). The scripting engines process the customer frameworks (“*API interfaces*”) and application scripts. De Castro, [0119]. For instance, a scripting engine—using the customer framework 1—“interprets App Script 1 and renders it in terms of the commands of the finite set defined at the API server” that are supported by the unified API (“*integrating, by the processor system, each of the plurality of API interfaces with a unified services platform API interface*”). De Castro, [0119].

109. De Castro further explains that 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.**” De Castro, [0119].

Further, “API client 630 then **packages the commands as messages** in accordance with the messaging API protocol.” De Castro, [0119].

110. In other words, the application scripts and customer frameworks hosted by the scripting engines are processed and 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. De Castro, [0118].

111. Fig.14 shown below illustrates scripting engines 600 and API clients 630 for processing the application scripts 610 and customer frameworks 620 into a finite set of commands supported by unified API 400. De Castro, Fig.14.



De Castro, Fig.14 (annotated)

112. 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 API interface*”). De Castro, [0127].

113. Indeed, a POSITA would have found it obvious that de Castro's system includes "*integrating [] the []API interfaces.*" For instance, integrating telecommunications services was well known in the art, as taught by Robb. Like de Castro, Robb describes "integrating a service specific to one of the plurality of the providers, and selectively presenting the specific service." Robb, 3:31-39. As another example, the Marascio (Ex.1010) reference also explains it was well known to integrate multiple APIs using a "provider framework" having a "well-defined API (application protocol interface) that enables third-party developers to build extensions"; Marascio also describes a "a management API that may be employed to integrate third-party network management tools." Maracio, Fig.2, 5:48-58, *see also* 5:31-47. In addition, Raleigh (Ex.1012) explains it was known for third parties to manage communication services for mobile wireless communication devices in a uniform consistent manner across different devices and/or different service providers using one or more APIs." Raleigh, 37:5-9, 458:5-22. A POSITA thus would have been aware of several techniques for "*integrating,*" for instance, as described by de Castro, Robb, Marascio, and Raleigh.

114. 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 API interface.”

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

115. De Castro’s “registry 520 [] registers ... the customers’ applications in the customer/developer domain 601.” De Castro, [0130]. Further, registry 520 also “log[s] ... the properties and status associated with each API clients 630.” De Castro, [0130]. That is, the registry logs information for each application and the API client of a scripting engine (“*service provider systems*”). De Castro, [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. De Castro, [0130].

116. A POSITA would have found it obvious for one of the “properties” stored in de Castro’s registry to be a short description or name of the application (each an example “*definition of a service*”). De Castro, [0130]. Storing a service’s name and/or description allows an administrator to determine the service to which the properties belong. The registry also logs “status” information for each of the applications (De Castro, [0130]), which a POSITA would have found obvious to include service availability information (another example “*definition of a service*”).

117. A POSITA thus would have found that these teachings in de Castro render obvious “*receiving a definition of a service ... for each of the plurality of*

service provider systems.” De Castro, [0130]. As discussed above at Section VII, the ’048 patent describes a service name, service description, and service availability as examples of “*a definition of a service.*” ’048 patent, 15:65-16:8. And de Castro’s disclosure of the registry including application properties (e.g., service name) and status information are like the examples of service name, service description, and service availability in the ’048 patent. De Castro, [0130].

118. Robb similarly describes “providing application services to customers” who use “SIP[] phones.” Robb, Abstract, 9:49-50. 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.” Robb, 7:29-32. For instance, Robb explains how the AIP facilitates the repackaging of services by compiling a “product catalog”:

The AIP 114 may also aggregate its own applications and services with those of both external ASPs/Internet Service Vendors (ISVs) and its own ASPs into a “**Product Catalog.**” The aggregate AIP Product Catalog may then be subscribed to by its ASPs 124.

Robb, 7:53-57

119. Robb further explains that the product catalog “may be used by new ASPs to select their services for customization and resale to customers.” Robb, 17:63-67. “The 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.” Robb, 26:47-49.

120. Thus, de Castro teaches registering each service application in a registry, including logging properties and status information of each application’s API client, and Robb teaches aggregating and publishing hosted services in a product catalog , which renders obvious “*receiving a definition of a service for each of the plurality of application service provider systems.*”

[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 with a telephone number;*

121. De Castro’s system includes “**a finite set of commands**” (“*one or more definitions of actions*”). De Castro, [0109]. De Castro explains that “an application program essentially issues a series of such **commands to operate the communication application server.**” De Castro, [0109]. 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.**” De Castro, [0118]. In a unified messaging protocol such as Rayo, “**a command carried in a message is received**” (“*receiving*”) at the communication application server 200. De Castro, [0122]. That is, the 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.”

De Castro, [0135].

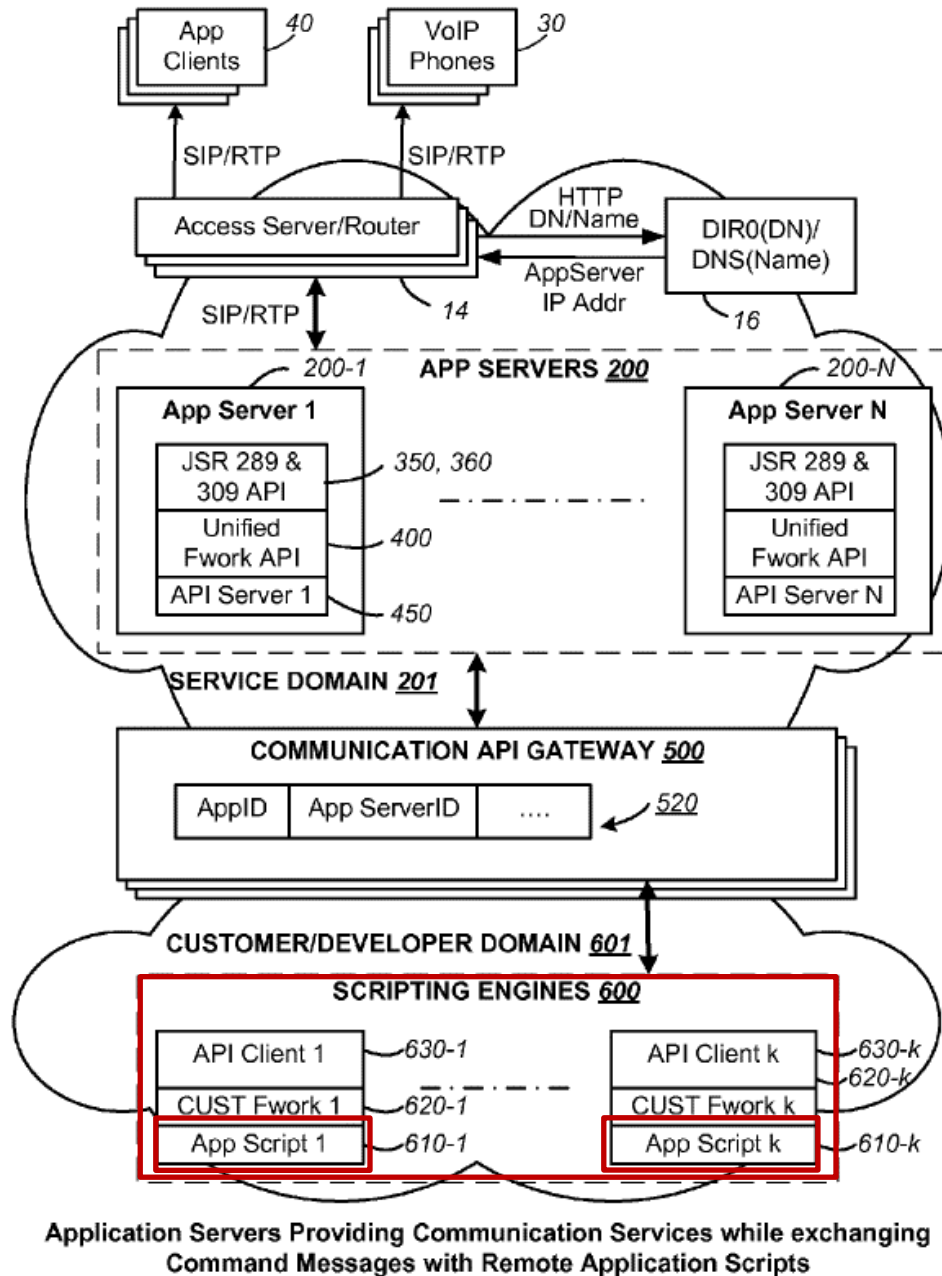
122. At the application communication server, “the API server **parses out the command in each message for execution in turn on the associate call actor.**” De Castro, [0124]. The parsed “commands” define the parameters of executable actions, such as to “**Answer**” or “**Redirect**” a telephone call. De Castro, [0141]. In this way, the commands of the application script, which are converted into Rayo by the API client, can be parsed and processed by the communication application servers, which can then control the phone call. *See* de Castro, [0118], [0124].

123. Therefore, de Castro’s communication application server receives commands for a service, where the commands define telephone call answer or redirect actions, which teaches “*receiving one or more definitions of actions for the service.*”

124. De Castro further explains that the parsed commands of the “Rayo protocol primarily deals with calls” to a telephone number. De Castro, [0139]. For instance, the command “**Redirect ... [o]nly applies to incoming calls.**” De Castro, [0141]. In this example, a call to the phone number “**to=tel:+18003211212**” can result in a redirect command “**redirect to=tel:+14152226789**” (“*action*”). De Castro, [0141]. This is an example

“triggering event occurring with a telephone number.” De Castro, [0141]. As another example, a call to a phone number where the “[d]ial destination is busy and cannot answer the call” (*“triggering event occurring with a telephone number”*) can result in an “<end/> [event that] indicates that the call has been disconnected” (*“action”*). De Castro, [0142]. The “redirect” or “disconnect” commands that execute, in response to a call to a phone number, render obvious that *“the actions comprise responses to triggering events occurring with a telephone number.”* De Castro, [0141]-[0142].

125. Regarding receiving definitions of actions for each service for each of the plurality of application service provider systems, Fig. 14 of de Castro illustrates multiple scripting engines 600.



De Castro, Fig.14 (annotated)

As discussed at [13.1], de Castro's scripting engines 600 correspond to *application service provider systems*. In the scripting engine 600 illustrated in Fig. 14, there are multiple applications scripts 610. Specifically, de Castro illustrates k application scripts as shown by App Script 1 (610-1) to App Script k (610- k). See de Castro,

Fig. 14. Each of these application scripts 610 corresponds to at least *one or more definitions of actions* in the form of a “finite set of commands.” De Castro, [0109]; *see also* De Castro, [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.”). 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*.

126. 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 with a telephone number.*”

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

127. De Castro explains how application scripts are developed. In particular, the “customer/developer domain 601” includes “a plurality of scripting engines 600.” De Castro, [0119].

[A]pplication scripting is decoupled from the operation of the communication application server, which only needs to focus on

providing basic communication services. A customer using the communication services can code the application script in a preferred programming language using a custom framework providing a set of customer-specific libraries. Scripting can be performed by third party scripting engines. The resulting execution codes only need be expressed in terms of the finite set of commands and sent as messages to operate the communication application server.

De Castro, [0041].

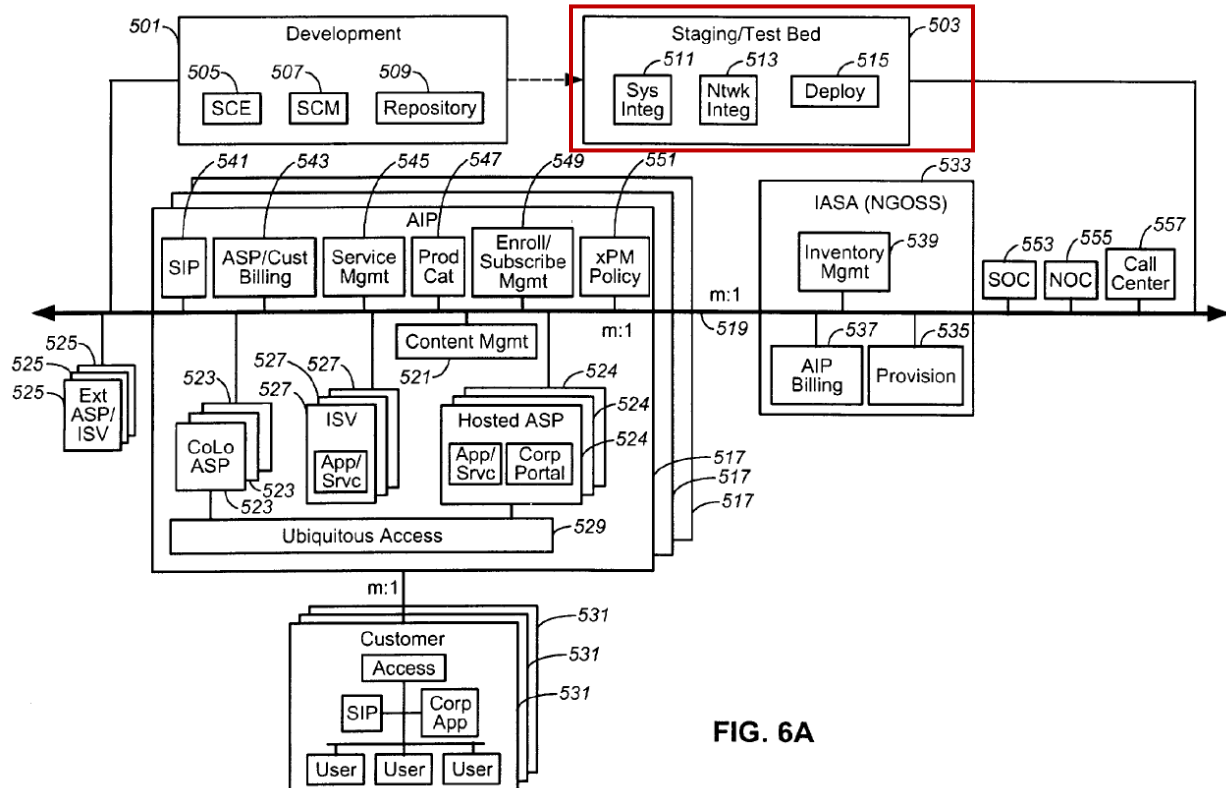
128. De Castro's application scripts outline specific "triggering events," such as incoming or outgoing calls, and their corresponding "actions," such as rerouting an incoming call to a different number. *See* De Castro, [0141]-[0142].

129. Robb also describes testing. In particular, Robb describes a "Staging and Test Bed environment." *See* Robb, Fig. 6A. A staging and test bed environment, like the one described in Robb, was used to test applications themselves as well as how those applications worked with other applications or functions within the broader environment in which the application was to be deployed. This is described in Robb.

The Staging and Test Bed system 503 includes System Integration or System Integrator (SI) 511, Network Integration 513, and Deployment 515, all of which may be located at different sites. 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**.

Robb, 17:2-9. The applications in Robb are integrated within the broader environment through triggering events and actions.

130. Regarding testing applications, Robb provides a “Staging/Test Bed” 503.



Robb, Fig. 6A (annotated)

131. Robb describes “a method for providing services and content over a network” including “presenting services including application level services.”

Robb, 3:29-32. In Robb, prior to deploying an application it is tested in a “test bed environment for conducting produce and service development.” Robb, 4:24-25; *see*

also Robb 23:38-40 (development environment/test bed “1021 supports development, **testing**, and integration of new services.”).

132. As taught by Robb, applications or services provided to users at the time could be integrated with each other through triggering events and actions, such that one application and service may work together with other applications or services. Robb, as well as the '048 patent, use a rules engine to implement these triggering events and actions. In Robb, the rules engine is part of an integration container.

In an exemplary embodiment, the Integration container 107 is supported by Enterprise Application Integration (EAI) technologies, which address integration of applications using messaging technology, including a message bus, a Workflow Manager (WFM), and **a Rules Engine (RE)**.... According to one feature of the present invention, the sequential rules of a process, or Workflow Management (WFM), are defined and stored centrally in the Rules Engine instead of embedded in code throughout the system. Work Flow Management automates the sequence of actions necessary to complete a unit of work. A specific work flow is often initiated when a rule is fired in a rules engine. **The Rules Engine 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 workflow**. Integration also includes version control of services within the execution environment and joining multiple directory servers by using a Meta directory.

Robb, 10:39-58. The “events” described in Robb function in the same way as the “*triggering events*” of the ’048 patent. The “specific workflow” in Robb that is initiated when an event occurs corresponds to the “*corresponding actions*” of the ’048 patent. Similarly, Robb describes its AIP system as a “prime example” of an “[e]vent driven system.” Robb, 27:30-31.

133. A POSITA at the time also would have understood how the events of Robb’s system are tied to the applications or services of the system and provides several example events which correspond to different types of applications, such as monitoring, time, health, or financial applications. *See* Robb, 27:34-37 (“In general, events can be triggered by applications, systems, and network elements, like error conditions, alarms, heartbeats, performance counter statistics, and billable events.”)

Application/System/Network Management tools ... use events to determine their health and availability. In addition, events may be used to determine the health and availability of applications.

Robb, 27:38-42; *see also* Robb, 28:14-16 (“SLA Management 1631 uses application/system/network events to determine service availability and delivery compliance.”). It would have been clear to a POSITA at the time, based at least on the disclosure of Robb, that Robb’s applications are associated with various events, such as those described above, and corresponding actions when the rules engine determines that the criteria is met. *See* Robb, 10:39-58.

134. A POSITA at the time would have also understood, based on Robb's disclosure of testing applications, events associated with those applications, and testing integration of applications, that testing an application would include testing the "*triggering event[s] and corresponding action[s]*" of that application. *See* Robb, 8:42-47, 31:22-23, 32:66-67. In particular, the AIP 114 "provides the core for 'gluing' services together." Robb, 8:45. The AIP 114 is "associated with" the "development and test bed environment." Robb 4:22-24. At least because integration includes monitoring events and executing corresponding actions, as explained previously (*see* Robb, 10:39-58), a POSITA would also have understood at the time that when Robb describes a "**tested and integrated environment**," the events and corresponding actions of applications are tested prior to deployment to the AIP 114. Robb, 8:42-47. Robb also explains that the "Test Bed 2107" is "where **integration** and tests are executed." Robb, 31:22-23. Robb also makes it clear that both testing of the individual applications is performed, as well as "verification tests" to "prove that the AIP is fully functional." Robb, 32:66-67. According to Robb, application and service integration involves verifying the correct functioning of applications with other applications via the "Rules Engine," which executes the specified triggering events and corresponding actions, through comprehensive testing. *See* Robb, 7:65-8:3 ("The AIP 114 integrates applications and services together by binding them ... through the use of ... Rules Engines

(REs).”). Robb also explains “[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.**” Robb, 17:5-7.

The Proprietor, through the AIP 114, may act as an “Application Infrastructure Provider” by offering a **tested and integrated environment** with an infrastructure that supports the technical and business processes that underlie ASP services. The AIP 114 provides the core for “gluing services together, and **offering common services and applications that can be used in conjunction with each other and ASP-specific capabilities.**

Robb, 8:41-48.

135. Thus, Robb’s description of testing “new applications” in the Staging/Test Bed 503 and performing verification tests includes testing corresponding events and workflows and corresponds to “*testing ... triggering event[s] and corresponding action[s] for ... service[s] in a test environment*” as claimed.

136. During prosecution of the application for the ’048 patent, the Patent Office issued a rejection of the claims as indefinite due to some terms related to testing. In their response, the applicant quoted a portion of the ’048 specification relating to the test environment.

The 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.

'048 Patent, 14:65-15:1. The applicant also explains, in its response, that “the test environment ... allows the application service providers to test the services and their interactions with the unified service platform.” Office Action Response dtd August 17, 2017, 12. But, the applicant also explains that “the test environment is the same as the unified services platform without the actual external connections or communications.” Office Action Response dtd August 17, 2017, 12. As explained in more detail below, a POSITA at the time would have understood this explanation from the applicant to mean that the test environment of the '048 patent does not have connections to entities or networks outside of the unified services platform in the way that the unified services platform can communicate with external entities or networks. In other words, while the unified services platform interacts with the real world through “actual external connections or communications,” the test environment operates in a dummy or mock environment which replicates or mimics the behavior of the live environment as closely as possible, but does not interact with the live environment through “actual external connections or communications.” As explained hereafter, it would have been obvious to a

POSITA to test applications in a mock environment in this way so that applications in testing don't interfere with the live environment.

137. The description in Robb supports this understanding. All the illustrations of a test bed or test environment in Robb show the test bed as a separate entity from the rest of the system. This is shown in Fig. 6A, below.

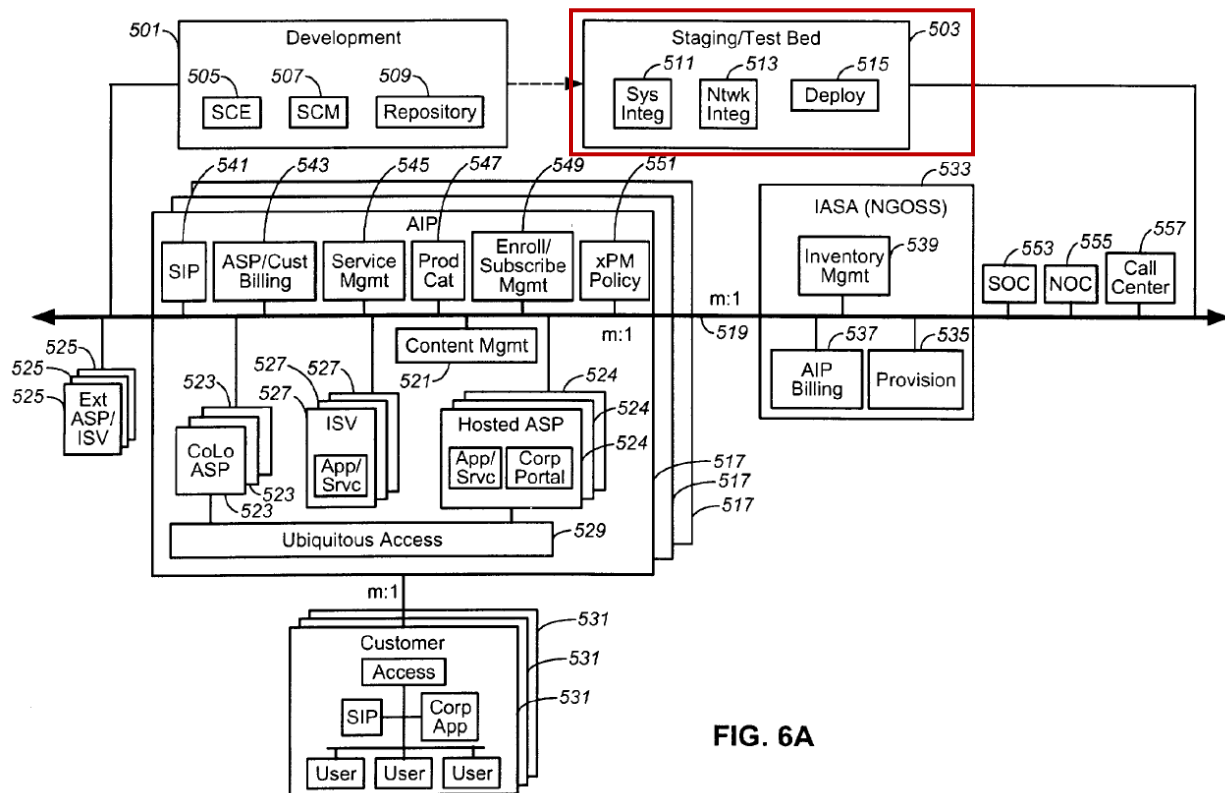


FIG. 6A

Robb, Fig. 6A (annotated).

In Fig. 6A, the Staging/Test Bed 503 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 receiving actual inputs from AIP 517,

or sending outputs to the AIP 517 (i.e., actual external communications or connections).

138. This separation of the test environment and the live environment is also depicted in Fig. 8, shown below.

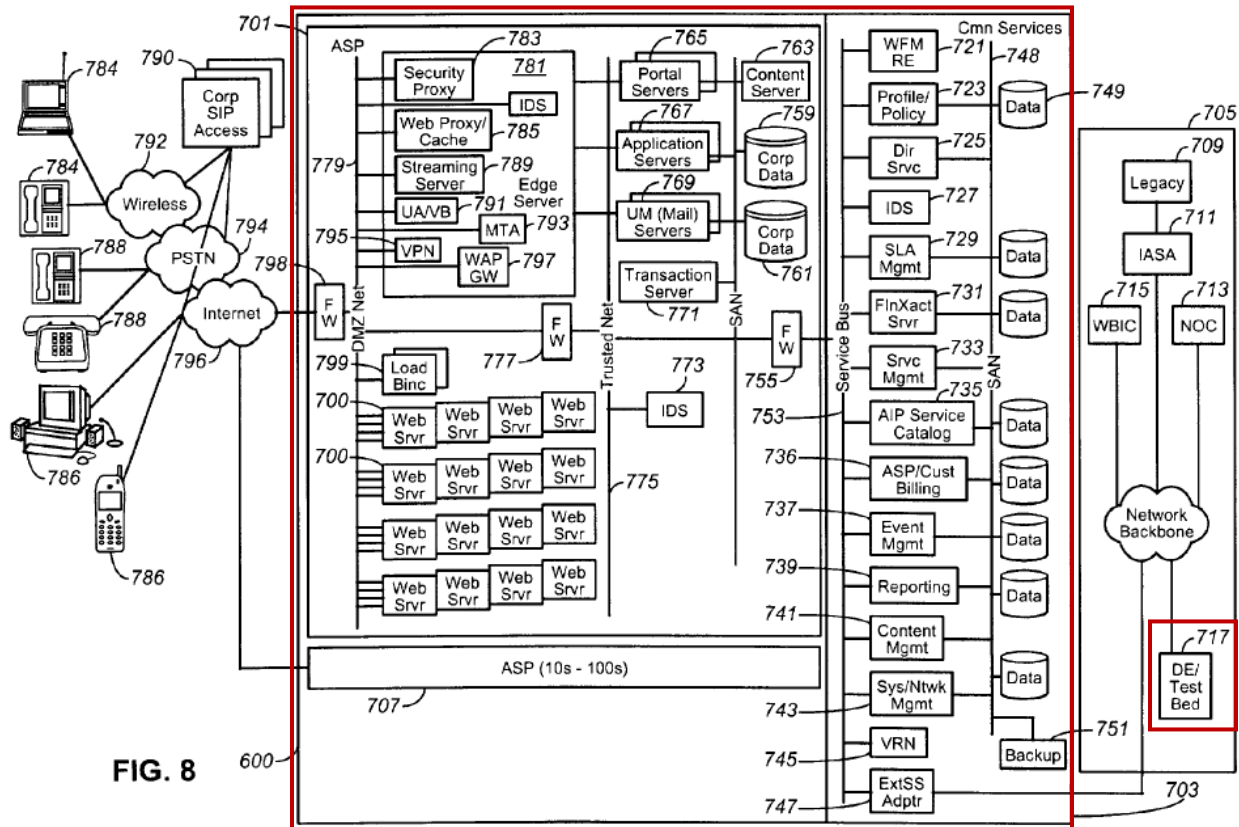


FIG. 8

Robb, Fig. 8 (annotated).

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.” Robb, 18:61-65. In Fig. 8, “the AIP is shown at 700.”³ Robb, 18:58. This phrase is likely

³ Robb at 18:58 recites “the AIP is shown at 700.” However, the reference numerals 700 in Fig. 8 point to “web servers.” Elsewhere, Robb describes Fig. 7 including a “(DMZ) Net 779” connected to “Load Balancers 799 and a

supposed to read “the AIP is shown at [6]00” because the reference number 700 in Fig. 8 is pointing to a set of web servers within an ASP 701. *See* Robb, Fig. 8. In addition, in describing Fig. 8, Robb identifies the “(DMZ) Net 779” connected to “Load Balancers 799 and a plurality of Web Servers 1100.” Robb, 19:42-44. This clearly shows that “the AIP is shown at 700” is a typo and should read “the AIP is shown at 600.” Fig. 8 illustrates the test bed 717 within the proprietor environment 705, which is separate from the AIP 600. *See* Robb, Fig. 8.

139. The understanding that the test bed described in Robb is the same as the AIP only without external communications is further supported by additional prior art. McCaffrey, for example, describes this exact situation as “Stub Testing and Mock Testing.” McCaffrey, 91. McCaffrey explains that “[s]tub testing is the process of creating a small, relatively simple placeholder method ... which can be called by the system under test.” McCaffrey, 91-92. “The term mock testing generally refers to a similar process where the placeholder is an OOP object and is created in part with the use of a code library.” McCaffrey, 92. McCaffrey then provides an example of “a system which makes airline flight reservations.” McCaffrey, 92.

A code snippet of a system which is designed using an OOP paradigm might look something like this:

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.”

```
Flight fit = new Flight("603");  
string set = "16e";  
Reservation res = new Reservation();  
res.Make(fit, seat);  
// code to verify state
```

The first statement creates a Flight object, and the third and fourth statements make a seat reservation. notice that in order to make a reservation, a Flight object must already exist. Now suppose you wish to test the method which makes a reservation but the Flight class has not yet been implemented. One solution is to create a simple stub code along the lines of:

```
public class Flight {  
    private string flightNumber;  
    public Flight(string flightNumber) {  
        this.flightNumber = "999";  
    }  
}
```

Here, the stub constructor merely **assigns a dummy value** to the flight number. Now a minimal **Flight class exists and software test engineers can call and exercise the Reservation.Make() method** to at least a minimal extent until the actual Flight class is implemented. In some testing scenarios stub code is created by testers with development skills, and in other scenarios testers make a request to the development to create stub code for the purposes of testing.”

McCaffrey’s example of testing software used to make flight reservations in which a stub system is created thus avoids creating actual flight reservations in the live system.

140. In addition, Desikan describes creating a mirrored environment for 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.**

Desikan, 136-37. Regarding integration testing, Desikan also describes the use of “stubs” to set up a test environment. *See* Desikan, 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.**”). Thus, a POSITA would have understood Robb’s “test AIP environment” (Robb, 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.” Office Action Response dtd August 17, 2017, 12.

141. Additionally, a POSITA at the time, based on the teaching of Robb, would have understood that testing applications would ideally be as comprehensive as possible, meaning that each application would ideally be tested, and each event and corresponding action associated with each application would also be tested.

For example, Fig. 28 in Robb illustrates a method whereby “an ASP add[s] service

packs to a corporate portal.” Robb, 34:16-17. “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.” Robb, 13:29-31. Thus, each time a new service is provided by an ASP, the steps of the method of Fig. 28 are performed, including testing. A POSITA at the time, therefore, based on Robb, would have understood that testing includes testing “*triggering event[s] and corresponding action[s] for **each** service in a test environment.*”

142. A POSITA would also have been aware that testing includes testing “*each triggering event and corresponding action for each service.*” A POSITA would have completed undergraduate coursework in computer programming as part of his or her bachelor's degree education in electrical engineering, computer engineering, or computer science. During this time, a POSITA would have gained knowledge of essential computer programming principles, such as the importance of comprehensively testing code.

143. The concept of testing software applications existed many years before the priority date of the '048 patent. Software testing has been performed for as long as programmers have developed code. A POSITA would also have known that an important aspect of testing a computer program was to test the software as comprehensively as possible. *See e.g.,* McCaffrey, 11 (“A full test pass suite generally consists of **every current test case.**”); Kaner, 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.”). Thus, 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.

144. A POSITA at the time would have been aware of the information provided in McCaffrey and Kaner. In particular, a POSITA would likely have encountered information like that provided in McCaffrey and Kaner in their undergraduate studies in, e.g., computer science, or in their professional experience. In that regard, McCaffrey describes his book as a “computer science book” intended for “software developers, project and program managers, students, [and] support engineers.” McCaffrey, 1-2. Similarly, Kaner describes his book as “for ... anyone that has to deal with testers.” Kaner, xxi. A POSITA would certainly have come into contact with and worked with testers during code development, either as an undergraduate student or during employment. In

addition, Kaner describes his book as “useful” to “programmers, project managers, and executives who have to work with testers.” Kaner, xxii.

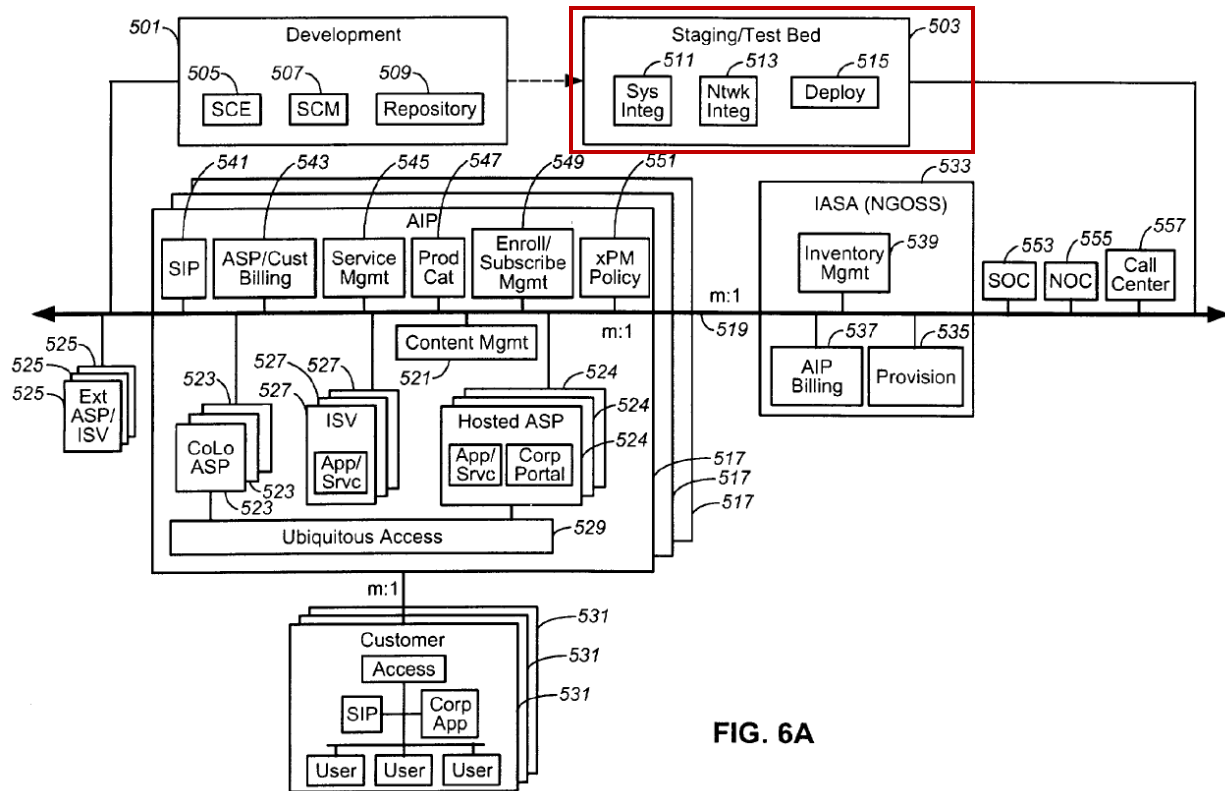
145. In addition, a POSITA at the time would have understood that testing in Robb would include “*generat[ing] a response*.” Testing at the time commonly involved comparing a response from code-under-testing to an expected response. This is also explained by McCaffrey: “the most common activity in software testing is **sending input** to the system under test and then **examining the result** to determine a pass or fail result.” McCaffrey, 5. 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.” McCaffrey, 5. Kaner also explains the prevalence of outputting a response as part of testing. Specifically, Kaner describes testing as “testers just execut[ing] test cases and **compar[ing] what happens against expected results**”). Robb’s “sequence of actions” and/or “specific work flow” initiated in response to a rule criteria being met is just one example of the outputs or responses Robb describes, which would be output during testing of those events and corresponding actions. *See* Robb, 10:39-58. It is my understanding that comparing software’s response under testing to an expected response was and is the most common method of testing. Therefore, a POSITA would have found it obvious for Robb’s testing processes to “*generate a response*” that would then be

compared to an expected response and used to evaluate whether the action was functioning correctly.

146. Based at least on the common place nature of the testing described in Robb, it would have been obvious to a POSITA to combine Robb's teachings regarding testing, including testing integration of applications via events and corresponding actions or workflows, to the teachings of de Castro. 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 Robb's testing method includes comprehensively testing computer programs. A POSITA would have been compelled to utilize Robb's testing principles to validate de Castro's application scripts and ensure their proper operation, given that deploying a computer program that doesn't function as intended is considered unfavorable. Thus, 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.

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

147. A POSITA would have understood that testing an application is to be performed prior to providing access to the application. For example, Robb explains that services are deployed only after testing is complete. Fig. 6A of Robb is reproduced below.



Robb, Fig. 6A (annotated)

As shown in Fig. 6A, the Staging/Test Bed 503 is positioned after the Development environment 501. Within the Staging/Test Bed 503, the steps of testing are shown, including system integration 511, network integration 513, and finally, after testing is complete, deployment 515. *See* Robb, 17:2-9. Robb also 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.**” Robb, 17:5-9. It would have been obvious to a POSITA that “*providing access to each service*” is done “*finally,*” in other words after, the “scaled system network test.”

Robb, 17:8. After testing, a new application is moved from the Staging/Test Bed environment 503 to the AIP. *See* Robb, 17:5-9. Robb clarifies that the tests are conducted in the Test Bed 503 before “deploying to AIPs,” providing further evidence that testing is carried out before deployment. Robb, 7:38-41. A POSITA at the time would have understood that the combination of de Castro and Robb teach “*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.*

148. As discussed above, de Castro teaches a registry that registers each application, including logging application “**properties and status**” information. De Castro, [0130]. 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. De Castro, [0130].

149. Additionally, “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.” De Castro, [0119]. That is, the application services are defined by “commands” coded in an “application script” (“*definitions of actions*”) that are understood by both the API client in the scripting engines

(developer domain) and the unified API in the communication application serves (service domain). De Castro, Fig.14, [0118], [0119], [0124].

150. Regarding the “*definition of the plan*,” 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**.” Robb, 26:67-27:5. As discussed above at Section VII, the ’048 patent describes service limits associated with the service—such as SLA terms and conditions—is an example “*definition of [a] plan*.” ’048 patent, 16:5-8.

151. It would have been obvious to combine Robb’s technique with de Castro’s system for the reasons provided above at Section IX.C. For instance, a POSITA would have been motivated to apply Robb’s product catalog (“*provisioning catalog*”) technique to de Castro’s system to allow service providers (e.g., de Castro’s developers) to repackage and bundle a wide variety of available application services to resell to their customers. A POSITA would have understood that de Castro’s developers (“customer/developer[s]”) are service providers, because de Castro explains that the developers use the scripting engines to “host and process [their] application scripts 610” for use by the app clients 40. De Castro, Fig. 14, [0119], [0116].

152. It would have been obvious to a POSITA that service providers would include standardized SLA terms or agreements (“*definition of the plan*”) that could be customized or “specified” by an AIP administrator, the service providers, and/or the end users. Robb, 33:39-42. For instance, as evidenced in the prior art, it was a common practice to provide template service agreements, or default SLA terms that could be customized or “superseded” by a service provider. For instance, Collins (Ex.1030) explains that an “End User Licence [sic] Agreement [EULA]” is “a contract between the User and the BCS Provider that specifies the **default terms** of subscription. It may be **superseded** by a Terms of Use Agreement for a particular Content Packager.” Collins, table at cols. 3 and 4. Similarly, Srinivas (Ex.1031) explains that

To expedite and/or simplify SLA negotiation, consumers may retrieve standard SLA templates from the provider. The provider may publish these standard SLA templates and describe the details of the various SLA templates, for example, conditions that apply to premium clients. Srinivas, [0058].

153. The combination, including the publishing of default or standard SLA terms, would simplify and streamline the service package publishing process by allowing ASPs to use the default SLA terms and modify as necessary. This would expedite the service provisioning process while allowing for service providers and

end users to specify the details of the agreements as needed, rather than define the SLA terms and criteria from scratch. Srinivas, [0058].

154. In the resulting combination, the properties and status of each application, such as a name or short description (“*definition of the service*”), as well as default service-level agreement (SLA) terms, conditions, and other SLA criteria (“[*definition of*] *the plan*”), would be included in information about the aggregated services that is published to or hosted by the AIP product catalog (“*provisioning catalog*”) for resale by ASPs. Thus, it would have been obvious to a POSITA that the AIP’s product catalog would publish “*the definition of the plan*” with each service added to the product catalog.

155. 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**. The aggregate AIP Product Catalog may then be subscribed to by its ASPs” who “select their services for customization and resale to customers.” Robb, 3:17-21, 17:65-67. “[C]ustomer ASPs may select from among numerous services in the AIP product catalog.” Robb, 26:24-26. The ASPs “repackage and resell [the selected services] to their customers as corporate portals.” Robb, 7:29-32.

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, Robb's product catalog would publish information about de Castro's available application scripts and associated commands (*"definitions of actions"*) for service providers to aggregate, repackage, and resell on their own corporate portals. Robb's 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. A POSITA would have found this combination obvious at least for the reasons provided above at element [13.4] and Section IX.C. 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."*

157. Robb teaches testing triggering events and corresponding actions with the "Staging and Test Bed Environment 503." Robb, 17:5-9. Robb uses the "Rules Engine (RE)," (Robb, 10:43) to "evaluate[] events using a rule base" and "initiat[e] a specific work flow" (Robb, 10:53-55). The '048 Patent uses a nearly identical "policy and rules engine" to identify "triggering event[s]" and "invok[e] ... service[s] ... in response to the ... triggering event[s]." '048 Patent, 46:43-48. These events and corresponding workflows are tested to ensure proper integration, as described at [13.6] above.

158. As explained at [13.7], Robb explains that services are provided after performing the test.

The Staging and Test Bed system 503 includes System Integration or System Integrator (SI) 511, Network Integration 513, and **Deployment** 515, all of which may be located at different sites. 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.

Robb, 17:5-9.

159. A POSITA would have found it obvious to combine Robb's teachings with de Castro's system at least for the reasons provided above at elements [13.3] and Section IX.C. In the resulting combination, a service provider accessible product catalog would publish, for each application, a service pack including the application script with its associated commands ("*definitions of actions*"), application properties and status information such as the name and a short description of the service ("*definition of the service*"), and default SLA terms and conditions ("*definition of the plan*"). 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 the user activates/subscribes to the service. *See* de Castro, [0022] (describing loading a Java script into processor

memory for processing). Therefore, a POSITA would understand that de Castro in view of Robb renders obvious performing the “*publishing*” step “*after testing*.”

160. Thus, 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, and Robb teaches aggregating and publishing services to a product catalog. Robb also teaches service providers selecting and customizing combinations of services as service packs, and defining SLA terms and conditions for the service packs. In combination renders 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*.”

2. Claim 14

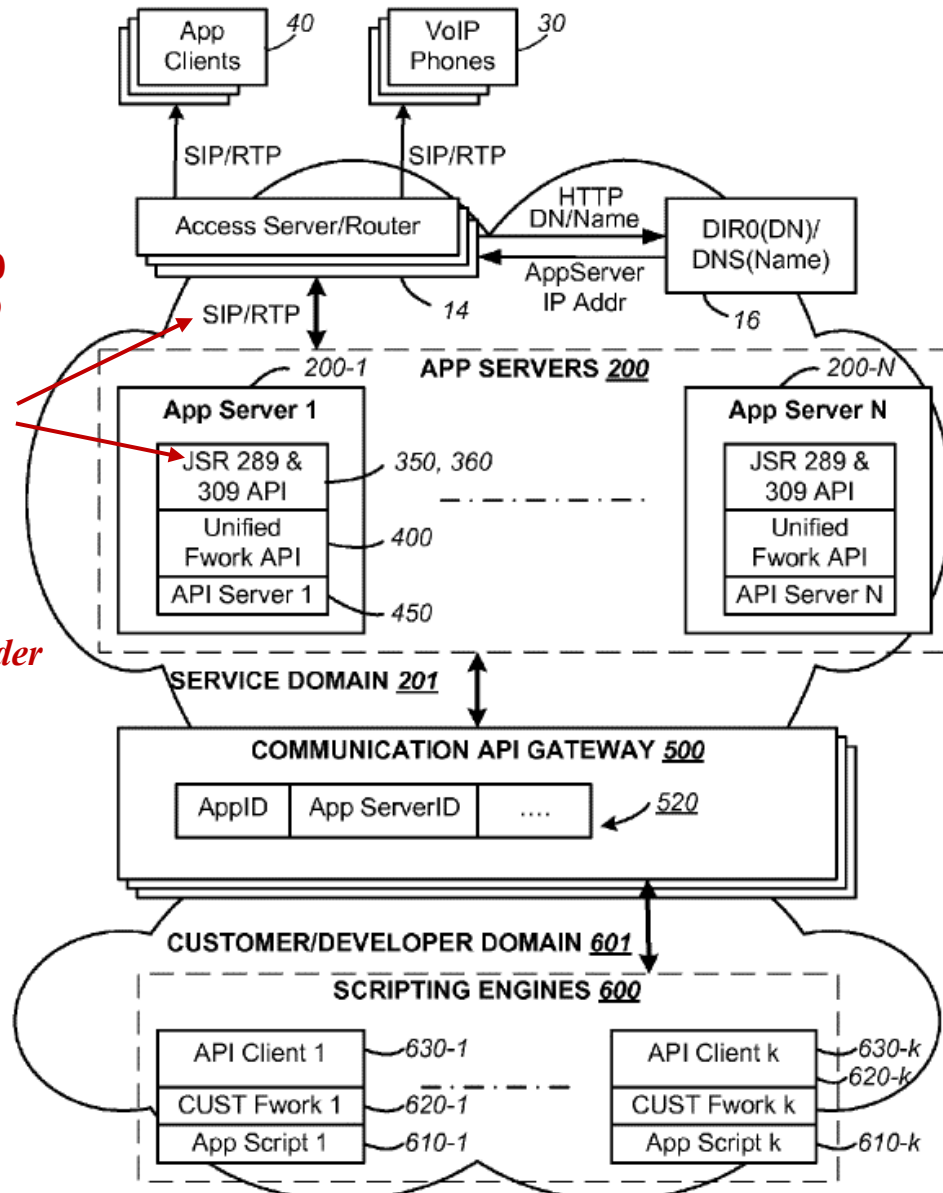
[14.1] *The method of claim 13, further comprising: receiving an API interface for a network provider system; and*

161. As discussed above, clients access the call-related application services “**via SIP**” networks (“*network provider system*”) connected to the application communication servers. De Castro, [0139], Fig.14.

162. The application communication servers 200 include a “**SIP Servlet (JSR 289) API 350**” (“*a provider API interface*”). De Castro, [0070]. As shown below in Fig.14, SIP API (JSR 289) 350 executes on communication application server 200 for interfacing with a SIP network. De Castro, Fig.14, [0070], [0139]. It

would have been obvious to a POSITA the SIP API was loaded onto or
“receiv[ed]” by server 200 prior to execution. See de Castro, Fig.14, [0070],
[0139]. Indeed, Fig. 14 illustrates SIP API 350 as being included within, and thus
already “receiv[ed]” by, server 200. See de Castro, 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
system”)**



Application Servers Providing Communication Services while exchanging
Command Messages with Remote Application Scripts

De Castro, Fig.14 (annotated)

163. 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.*”

[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.

164. The “JSR 289 API 350” (e.g., SIP API) is a programming interface within the communication application servers that “deal[s] with high-level objects abstracted from low-level” SIP network communication objects, for use by the unified framework AIP 400. De Castro, [0070], [0074], Fig.14.

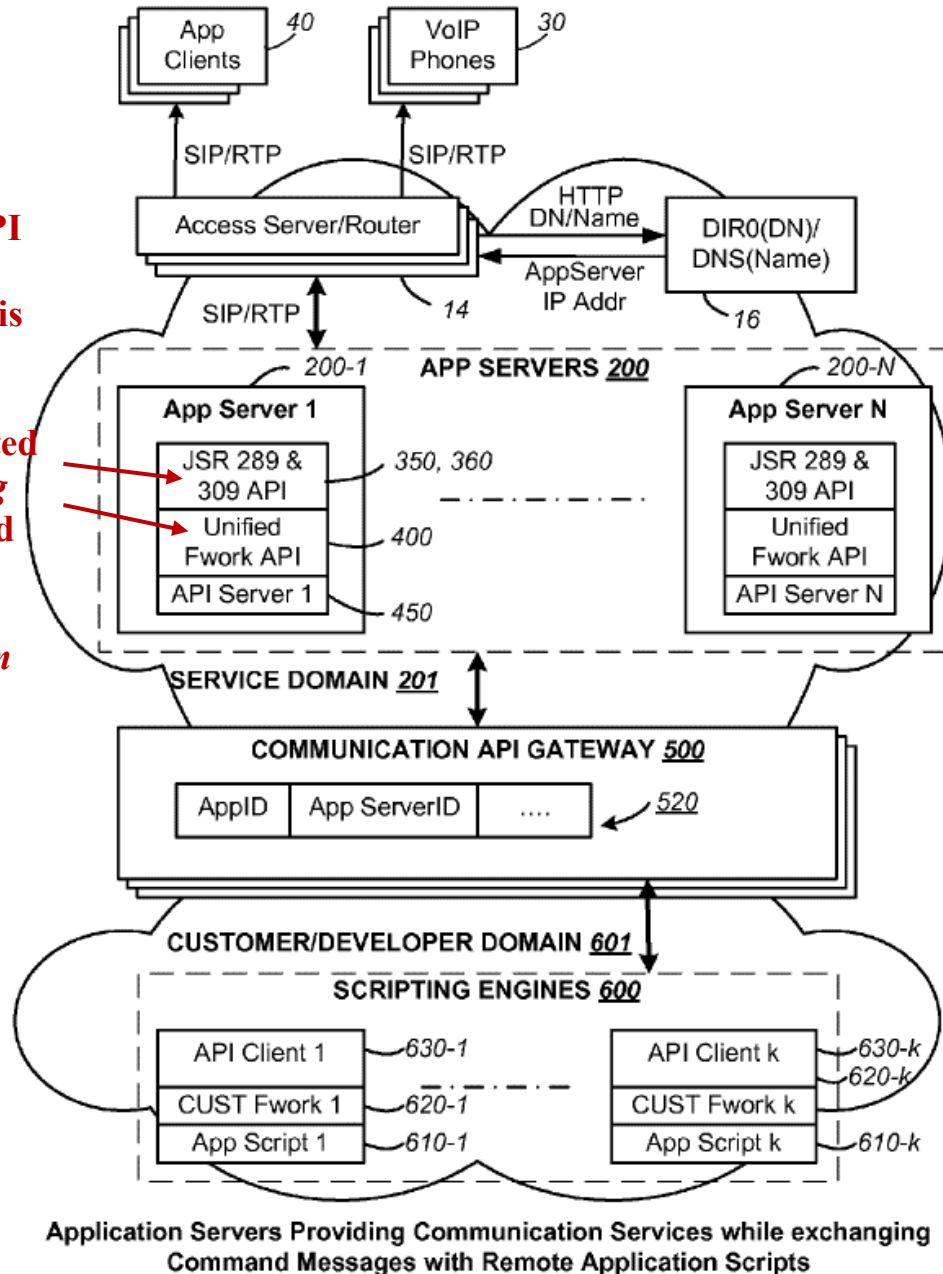
165. The “unified communication framework provides high-level, unified objects built from lower-level object of JSR 289/309.” De Castro, [0096].

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.” De Castro, [0074].

166. As shown in Fig.14 below, JSR 289 (SIP API 350) interfaces with unified framework API 400. De Castro, 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. Therefore, a POSITA would have understood this integration of JSR 289 with the unified API 400 to teach “*integrating the provider API*

interface with the unified services platform API interface,” as claimed. De Castro, Fig.14, [0096].

JSR 289 SIP API 350 (“provider API interface”) is interfaced with and provides objects abstracted for (“integrating ... with”) unified framework API 400 (“unified services platform API interface”)



De Castro, Fig.14 (annotated)

167. Further, 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. De Castro, [0067], [0070], [0041], [0106], [0119].

168. De Castro explains that “**SIP servlet (JSR 289) for call control ... provides a Java API**” (“*provider API interface*”). De Castro, [0026]-[0027]. Here, Java refers to a common type of programming language. De Castro, [0026]. De Castro further explains that “SIP servlets interact with clients by **responding to incoming SIP requests and returning corresponding SIP responses.**” De Castro, [0026]. The JSR 289 SIP API, therefore, deals with “**low level call control objects**” of an SIP network. De Castro, [0030], [0070].

169. 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. De Castro, [0026], [0106], [0119]. 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.**” De Castro, [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.**” De Castro, [0106], [0041].

170. Therefore, a POSITA would have understood that the Java-based JSR 289 SIP API 350 is not compatible with the API of an application script coded in, e.g., Ruby or Python. De Castro, [0070], [0106]. For instance, the SIP API is not natively compatible with the Rayo API, as Rayo messages cannot be parsed by a SIP server. 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 provider API interface.*”

171. Indeed, a POSITA would have found it obvious that de Castro’s system includes “*integrating the provider API interface.*” For instance, integrating telecommunications network provider interface was well known in the art, as taught by Robb. Like de Castro, Robb generally relates to “providing comprehensive services including integrated Application Service Provider (ASP), Independent Service Vendor (ISV), and the like services, software and hardware.” Robb, 1:17-20. Robb further describes it was known to use a “platform [] configured to integrate the specific service and to selectively present the specific service associated with the one provider to another one of the plurality of providers.” Robb, 3:47-50. Indeed, it was common to use a testing environment or “a Test Bed to integrate the solutions” such as multiple APIs into a unified API. See Robb, 7:38-46.

172. As another example, the Marascio (Ex.1010) reference also explains it was well known to integrate multiple APIs using a “provider framework” having a “well-defined API (application protocol interface) that enables third-party developers to build extensions”; Marascio also describes a “a management API that may be employed to integrate third-party network management tools.” Maracio, Fig.2, 5:48-58. In addition, Raleigh (Ex.1012) explains it was known for third parties to manage communication services for mobile wireless communication devices in a uniform consistent manner across different devices and/or different service providers using one or more APIs.” Raleigh, 37:5-9; *see also* Raleigh, 359:39-57 (“[O]ne or more device agents in the mobile device 100 are integrated with one or more communication stacks, e.g., as illustrated in FIGS. 9 to 17. In some embodiments, communication between elements in the communication stacks is realized using one or more APIs.”), 458:5-46 (“many different types of services can be supported through the one or more APIs [] including application specific services”). A POSITA thus would have been aware of several techniques for “*integrating*,” for instance, as described by de Castro, Robb, Marascio, and Raleigh.

173. 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 provider API interface with the unified services platform 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 provider API interface.*”

3. 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.

174. De Castro teaches a service that processes call control commands and routes the service to a subscriber’s phone number. De Castro also provides an example where a communication application 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 execute the specific communication application.” De Castro, [0116]. For the incoming call either to or from the client’s 40 telephone number (“*a communication event with the telephone number*”), the directory DIR0 routes the call to/from the client and the server that executes the communication application (“*invoking the [] service*”). De Castro, [0141]. As a result, the server “can now control the call by using ... commands,” and the actions of the commands can be routed to the client 40. De Castro, [0141]; *see also id.* [0007] (discussing activation

of various services, such as “Follow me, find me,” in response to inbound calls).

Thus, de Castro teaches routing commands to a communication application server for execution in response to an inbound call (“*triggering event*”) from a client’s telephone number. De Castro, [0116], [0141].

175. De Castro further explains that the inbound call uses Session Initiation Protocol (SIP) (“*the triggering event comprises ... a Session Initiation Protocols (SIP) on receiving a call*”). De Castro, [0141]. For instance, “[i]nbound calls originate from the PSTN or **via SIP.**” De Castro, [0139]. Indeed, Fig.14 of de Castro (reproduced above), illustrates clients 40 (e.g., subscribers) connected to the communication application servers via an SIP/RTP network(s). De Castro, Fig.14. 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. -->
```

De Castro, [0141].

176. 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. De Castro, [0117], [0141].

177. 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.*”

X. CONCLUSION

178. I declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true, and that these statements were made with knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under section 1001 of Title 18 of the United States Code.

Dated: 4/3/23

Henry Houh
Henry Houh, Ph.D.