

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

MICROSOFT CORPORATION
Petitioner,

v.

EDGE NETWORKING SYSTEMS, LLC,
Patent Owner.

U.S. Patent No. 11,695,823
Title: DISTRIBUTED SOFTWARE DEFINED NETWORKING

Inter Partes Review No.: IPR2025-00618

**DECLARATION OF DR. SANJAY RANKA IN SUPPORT OF
PATENT OWNER'S RESPONSE**

Table of Contents

- I. Introduction And Qualifications 1
- II. Summary Of Materials Reviewed And Considered 6
- III. Legal Standards 7
- IV. Level Of Skill in the Art and Perspective Applied in This Declaration 9
- V. The Specification of the '823 Patent 11
 - A. Overview Of Distributed Application Environment Disclosed In '823 Patent..... 11
 - B. Virtual Fabric For Communicating Between The FxCloud App Component and FxDevice Component Of A Distributed Application..... 14
 - C. Software Layers Of Platform Architecture..... 15
 - D. APIs For Communications Between Different Distributed Application Or With Entities Outside Of The dSDN Platform 18
 - E. Examples Of Upper Layer APIs Disclosed In The '823 Patent 19
 - F. Explicit API Frameworks And Platform APIs 21
 - G. The Achievements Of The '823 Patent 24
- VI. Claim 1 Of The '823 Patent..... 29
- VII. STATE OF THE ART - JAVA DEVELOPMENT ENVIRONMENT 38
- VIII. Prior Art Relied On In Petition 41
 - A. Vasell 41
 - B. Alves 49
 - C. Hall..... 50
 - D. Rellermeyer..... 52
- IX. Overview Of Distinctions Between the '823 Patent And Vasell 53

X. Distinctions Between "Upper Layer API" As Set Forth In Claim 1
Of the '823 Patent And Vasell..... 56

XI. Distinctions Between "Unified Capabilities" As Set Forth In Claim 1
Of the '823 Patent And Vasell..... 67

XII. Certification..... 83

I. Introduction And Qualifications

1. I, Sanjay Ranka, submit this declaration on behalf of Edge Networking Systems, (“Patent Owner”) to provide my opinions concerning the validity of claims 1-5, 7-8, 12-15, and 18-19 (the “Challenged Claims”) of U.S. Patent No. US 11,695,823 (the “ ‘823 patent,” Ex.1001), in support of Patent Owner’s Response to the Petitioner for *Inter Partes* review of the ‘823 patent, which I understand was filed by Microsoft Corporation., (“Microsoft”, the “Petitioner”) and accompanied by a Declaration of Dr. Erez Zadok (the “Zadok Declaration”), among other exhibits.

2. I am more than eighteen years of age, and I am a citizen of the United States, currently residing in the State of Florida.

3. In formulating my opinions, I have relied upon my knowledge, training, and experience in relevant art. My qualifications, along with a more detailed summary of my background and publications, are stated more fully in my *curriculum vitae*, which has been provided as Ex. 2004. Here, I provide a concise summary of my qualifications.

4. I am a Distinguished Professor in the Department of Computer Information Science and Engineering at the University of Florida.

5. Between 1999 and 2002, I served as Chief Technology Officer at Paramark, located in Sunnyvale, CA. During this tenure, I led the development of

a real-time optimization system known as PILOT (Paramark Interactive and Learning Optimization Technology), designed specifically for marketing campaign enhancement. PILOT achieved the delivery of over 10 million optimized decisions daily by 2002 using a parallel and distributed system, maintaining an operational uptime of 99.99% or higher. The PILOT system incorporated proprietary algorithms that continuously refined marketing campaigns through real-time data collection and performance analysis. The PILOT technology represents a significant advancement in adaptive marketing optimization based on dynamic data-driven decision making. Paramark's technological innovation was recognized by VentureWire and Technologic Partners, ranking the company among the Top 100 Internet technology firms in both 2001 and 2002, culminating in its acquisition in 2002.

6. I have also held positions as a tenured faculty member at Syracuse University, as an academic visitor at IBM, and as a summer researcher at Hitachi America Limited.

7. I have coauthored one book, five monographs, and 350+ journal and refereed conference articles. I have been recognized as a fellow of the Institute of Electrical and Electronics Engineers (IEEE), and the American Association for the Advancement of Science (AAAS) for my contributions to parallel and distributed computing. I was also a past member of International Federation for Information

Processing (IFIP) Committee on System Modeling and Optimization. Pertinent to the field of the '823 Patent, I won the 2020 Research Impact Award from IEEE Technical Committee on Cloud Computing. I was awarded the 2022 Distinguished Alumnus Award from Indian Institute of Technology, Kanpur.

8. I am currently an associate editor for CRC Press for Bigdata. In the past, I have been an associate editor for IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, ACM Computing Surveys, and IEEE/ACM Transactions on Computational Biology and Bioinformatics, Sustainable Computing: Systems and Informatics, Knowledge and Information Systems, and International Journal of Computing. I was an associate editor-in-chief of the Journal of Parallel and Distributed Computing for the last five years. I was a general co-chair for the International Conference in Data mining in 2009, the International Green Computing Conference in 2010 and the International Green Computing Conference in 2011, a general chair for ACM Conference on Bioinformatics and Computational Biology in 2012, and a program chair for 2013 International Parallel and Distributed Processing Symposium and 2015 High Performance Computing Conference. I was a co-general chair for DataCom 2017 and co-program chair for ICMLDS 2017 and 2018. I was a program co-chair for the International Conference in Data Mining for 2022. My work has received 18000+ citations with an h-index of 65 (based on Google Scholar). ScholarGPS

awarded me in 2024 with the Highly Ranked Scholar status for my contributions to parallel computing.

9. I am a named inventor on several patents and published patent applications corresponding to the areas of my professional work. Accordingly, I have a general understanding of the U.S. patent prosecution process and of the novelty and non-obviousness requirements for patentability. Examples of patents where I am named as an inventor include:

- United States Patent Number 8,805,715. Brian Jones and Sanjay Ranka. Method for Improving the Performance of Messages Including Internet Splash Pages, August 12, 2014.
- United States Patent Number 8,386,315. Aravind Bala, Richard Chatwin, Brian Jones, Ismail Ahmet Nalcacioglu, and Sanjay Ranka. Yield Management System and Method for Advertising Inventory, February 26, 2013.
- United States Patent Number 8,260,663. Sanjay Ranka, E. Diane Chang, and Daniel Veiner. Method, Algorithm, and Computer Program for Targeting Messages Including Advertisements in an Interactive Measurable Medium, September 4, 2012.
- United States Patent Number 7,415,423. Sanjay Ranka, Jason Lenderman, and James Weisinger. Method, Algorithm, and Computer Program for

Optimizing the Performance of Messages Including Advertisements in an Interactive Measurable Medium, August 19, 2008.

- United States Patent Number 7,406,434. E. Diane Chang, Richard Chatwin, Sachin Kumar, Sanjay Ranka, James Weisinger, and Jason Lenderman. System and Method for Improving the Performance of Electronic Media Advertising Campaigns through Multi-attribute Analysis and Optimization, July 29, 2008.

10. I believe that my extensive industry experience and educational background qualify me as an expert in the relevant field of parallel and distributed computing systems and cloud computing as described in the '823 patent and related art.

11. I am knowledgeable of the relevant skill set that would have been possessed by a hypothetical person of ordinary skill in the art ("POSITA") at the time of the claimed invention of the '823 patent, which I have treated as June 13, 2013 for purposes of this proceeding. I have the necessary expertise to provide my informed opinions in this IPR, and I am an expert in the art of the '823 patent, due to at least the following:

- I received my undergraduate, master's, and doctoral degrees in computer science and engineering, computer and information science, and computer and information science, respectively;

- I have extensive experience in the parallel and distributed computing. I have been conducting state-of-the-art research in the use of parallel, distributed and cloud computing for a variety of disciplines including scientific computing, transportation, and health care for the last three decades;
- I have been recognized by IEEE and AAAS as a fellow for my contributions in parallel and distributed computing;
- I was awarded the 2020 Research Impact Award from the IEEE Technical Committee on Cloud Computing for "significant and impact contributions to theory and practice of cloud computing,"
- I have extensive experience in software design and development in both academia and industry; and
- I have supervised 50+ doctoral students in computer science and related fields during the last 30+ years and many of these students are currently employed by companies such as Facebook, Amazon, and Microsoft.

II. Summary Of Materials Reviewed And Considered

12. The opinions contained in this Declaration are based on the documents I reviewed and my knowledge and professional judgment. In forming the opinions expressed in this Declaration, while drawing on my experience, I reviewed the Petition, the exhibits referenced in the Petition, the Patent Owner Response, and the exhibits referenced in the Patent Owner Response.

13. My opinions are additionally guided by my appreciation of how a person of ordinary skill in the art (POSITA) would have understood the claims of the '823 patent at the time of the invention, which I have been asked to assume is June 13, 2013. For purpose of this declaration, I will use the definition of POSITA as provided by Dr. Zadok (Ex 1003) and described in Section IV.

III. Legal Standards

14. I am not an attorney. However, I have been informed by counsel of certain legal standards and have applied those standards, described below, in formulating my opinions.

15. I understand that the claims of a patent define the limits of a patentee's exclusive rights. I understand that to determine the scope of the claimed invention, courts may construe claim terms, the meaning of which the parties may dispute. I understand that claim terms should generally be given their ordinary and customary meaning as understood by a person of ordinary skill in the art at the time of filing of the patent application.

16. I understand that claims must be construed in light of and consistent with the intrinsic evidence. In this context, I understand that intrinsic evidence includes the claims themselves, the written disclosure in the specification, and the patent's prosecution history. I understand that the specification is always highly relevant to the claim construction analysis and often is the single best guide to the

meaning of a disputed term. I understand that extrinsic evidence may also be considered when construing claims and may include, for example, technical dictionaries, technical publications and books, treatises, and expert testimony, though the intrinsic record will be more persuasive as to the meaning of a particular term.

17. I understand that one or more prior art references can render a patent claim obvious to a person of ordinary skill in the art if the differences between the subject matter set forth in the patent claim and the prior art are such that the subject matter of the claim would have been obvious at the time the claimed invention was made. In analyzing obviousness, I understand that it is important to consider the scope of the claims, the level of skill of a person of ordinary skill in the relevant art, the scope and content of the prior art, the differences between the prior art and the claims, and “secondary considerations” of non-obviousness, which I also understand can include, for example, evidence of commercial success of the invention, evidence of a long-felt need that was solved by an invention, evidence that others copied an invention, or evidence that an invention achieved an unexpected result.

18. I understand that the claimed subject matter can be obvious when a person of ordinary skill in the art would have been motivated to combine pre-existing elements, and that combination yields no more than what a person of

ordinary skill in the art would have expected. I also understand that, in assessing whether a claim is obvious, one must consider whether the claimed improvement is more than the predictable use of prior art elements according to their established functions. I understand that there need not be a precise teaching in the prior art directed to the specific subject matter of a claim because one can take account of the inferences and creative steps that a person of ordinary skill in the art would employ. However, I further understand that obviousness cannot be based on the hindsight combination of components selectively culled from the prior art.

19. I understand that for a reference to qualify as a relevant prior art reference in an obvious analysis, the reference must be “analogous”—meaning it is from the same field of endeavor as the patented technology, and if not, then the reference must be reasonable pertinent to the particular problem facing the inventors of the patent at issue. I also understand that a reference is reasonably pertinent if a person of ordinary skill in the art would have reasonably consulted those references and applied their teachings in seeking a solution to the problem facing the inventors.

IV. Level Of Skill in the Art and Perspective Applied in This Declaration

20. I understand that certain issues relating to validity must be judged from the perspective of a person of ordinary skill in the art (“POSITA”). I further understand that factors that may be considered in determining the level of skill of

a person of ordinary skill in the art include, but are not limited to, (1) the type of problems encountered in the art; (2) prior art solutions to those problems; (3) rapidity with which innovations are made; (4) sophistication of the technology; and (5) the educational level of active workers in the field.

21. I have been informed and understand that a person of ordinary skill in the art is a hypothetical person who is presumed to be aware of all pertinent prior art, thinks along conventional wisdom in the art, and is a person of ordinary creativity. Importantly, this hypothetical person is of ordinary skill in the art to which the claimed subject matter pertains.

22. With respect to the '823 patent, as per Dr. Zadok declaration, "POSITAs as of June 13, 2013 would have had a bachelor's degree in computer science, computer engineering, or equivalent degree, and approximately three years of experience working in the computer science or engineering field. Additional experience might substitute for less education and vice versa. To that end, POSITAs in June 2013 would have been knowledgeable about the design and management of networked systems and virtualization technologies, and familiar with operating/distributed systems and security and privacy techniques." Ex. 1003, ¶58.

23. I agree with Dr. Zadok, and have applied the above-defined perspective of a person of ordinary skill in the art, as of June 13, 2013, in forming

and expressing my opinions. Unless otherwise stated, my testimony below refers to the knowledge of one of ordinary skill in the art at that relevant time.

24. At the time of the earliest priority date for the '823 patent (June 13, 2013), I met the qualifications to be a POSITA, and I believe that I also have a sufficient level of knowledge, experience, and education to provide expert opinions in the field of the '823 patent.

25. While my own level of skill exceeded that of the ordinary level of skill as of June 13, 2013, I am well-acquainted with the actual performance and capabilities of a person of ordinary skill in the art as defined above. For example, during the relevant timeframe, I was supervising and evaluating the performance of such individuals, such as my graduate students who were performing research in this area. Furthermore, my participation in professional organizations and in conferences and other events exposes me to the state of the industry and the level of experience of individuals who, like I did, had at least the ordinary level of skill.

V. The Specification of the '823 Patent

A. Overview Of Distributed Application Environment Disclosed In '823 Patent

26. The '823 Patent discloses a “Distributed Software Defined Network (dSDN)” which “is an end-to-end architecture that enables secure and flexible programmability across a network with full lifecycle management of services and applications (fxApp).” Ex. 1001, 7:57-61. “The dSDN also harmonizes fxApp

deployment across the network independent of the hardware vendor." *Id.*, 7:61-63.

"In the dSDN architecture, the network features are virtualized and may be distributed across separate network elements that cooperate together to create an advanced, programmable, and scalable network." *Id.*, 10:2-6.

27. "A high-level overview of a dSDN system 300 is depicted in FIG. 3," which is reproduced with colored annotations added. *Id.*, 10:8-9. The dSDN system 300 includes, *inter alia*, "a flexible network device (fxDevice) 302 [red box]" and "a flexible cloud platform (fxCloud) 304 [green box]." *Id.*, 10:9-11.

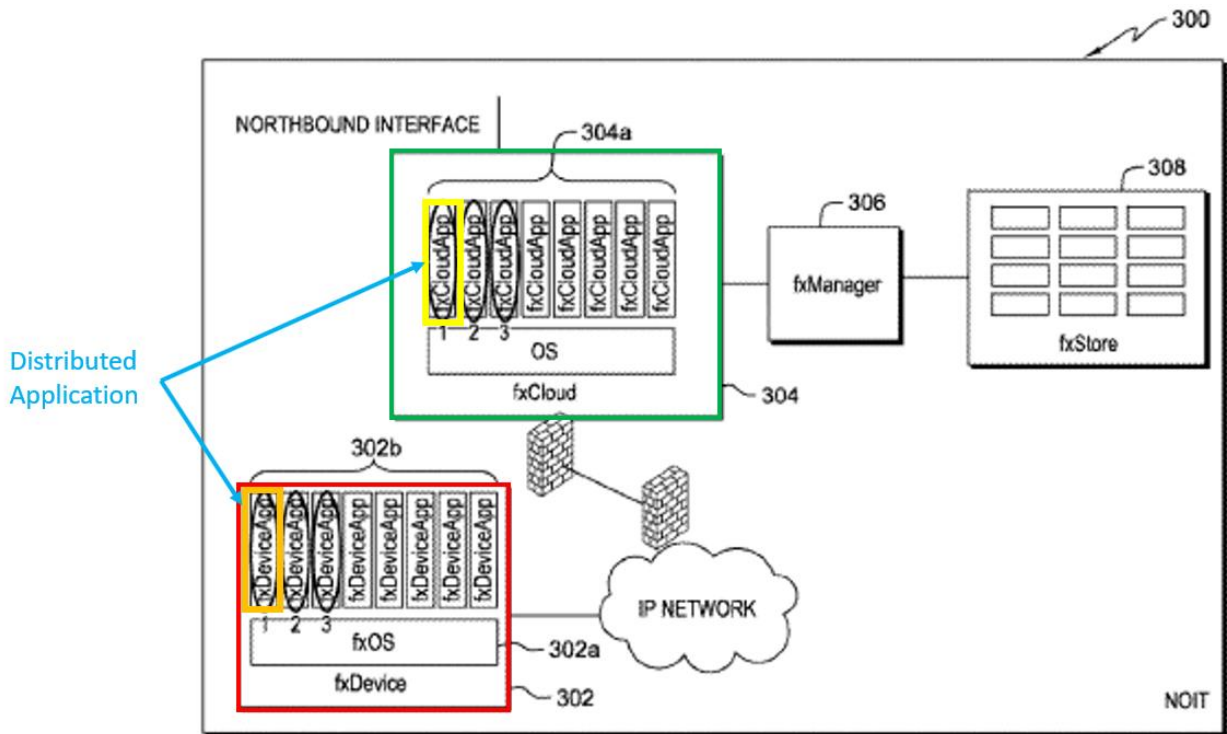


FIG. 3

28. In the dSDN system shown above, a *FxDeviceApp 302b* [orange box] securely communicates with a *fxCloudApp 304a* [yellow box] to form a distributed application [identified with blue arrows and text]. *Id.*, 1:39-41.

29. The '823 Patent explains that a "distributed application" is formed from the combination of an *fxDeviceApp 302* together with its "sister app" in the backend cloud infrastructure and referred to as *fxCloudApp 304a*:

In the dSDN system, *fxDeviceApp 302b* may have a sister app in the backend cloud infrastructure (i.e., flexible cloud platform 304) referenced here as *fxCloudApp 304a*. The *fxCloudApp 304a* in the cloud is paired with its *fxDeviceApp 302b* in the *fxDevice 302*. The *fxCloudApp 304a* and the *fxDeviceApp 302b* collectively form a distributed application (*dApp or fxApp*). In this description, when *fxDeviceApp* is referenced it shall mean any application that may have software components in the *fxDevice*, *fxCloud*, or both. In general, an *fxApp* package may include the following software components:

fxDeviceApp binary;
fxCloudApp binary;
Manifest files; and
Signatures.

It is important to note that the *fxDeviceApp 302b* and the *fxCloudApp 304a* *could use any protocol to communicate with each other and there is no need for standardizing this communication allowing for ultimate freedom for the developers*. This allows for the system to operate in a loosely coupled autonomous fashion allowing for asynchronous communication in a distributed fashion. Ex. 1001, 10:25-46 (emphasis added).

30. Thus, in the context of the '823 Patent, a distributed application comprises an *fxCloudApp 304a* in the cloud that is paired with its *fxDeviceApp*

302b in the fxDevice 302. In addition, as explained next, the platform uses a virtual fabric (fxVF) that enables the exchanges of messages between the fxCloud portion and the fxDevice portion of a distributed application and between different distributed applications. *Id.*, 13:33-36. The virtual fabric functions as an abstraction layer that hides the complexities of messaging from the developers. *Id.*, 13:36-37.

B. Virtual Fabric For Communicating Between The FxCloud App Component and FxDevice Component Of A Distributed Application

31. The dSDN architecture includes a virtual Fabric (fxVF) that "enables transparent switching of application and system messages" between the fxCloud portion and the fxDevice portion of a distributed application (dAPP) and between different dApps. *Id.*, 13:33-36. The virtual fabric "is an abstraction layer that hides most of the complexities of messaging from the developers." *Id.*, 13:36-37. An example of operation of the virtual fabric [red boxes] in the environment of an fxCloud and fxDevices is depicted in Figure 15 (reproduced below with colored annotations added). *Id.*, 2:23.

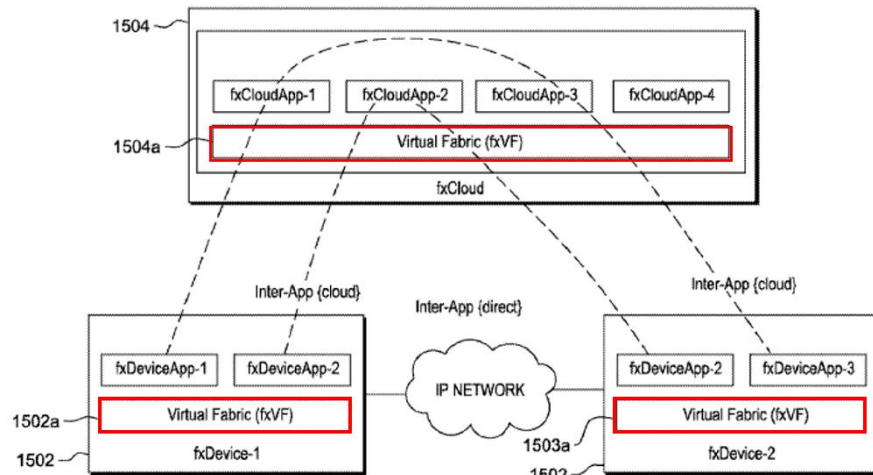


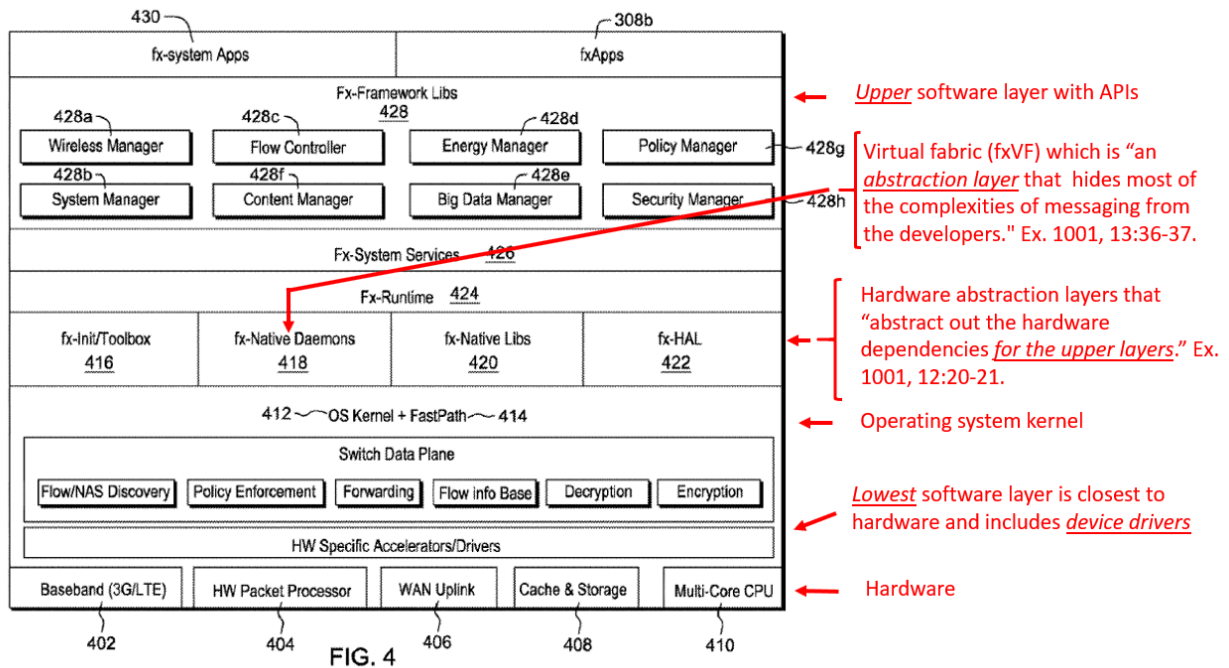
FIG. 15

32. The primary function of the virtual fabric is to hide the complexity of communications between different components of a dApp or between dApps and make such complexities abstract for the developer." *Id.*, 11:50-61 , 13:33-36 (the virtual fabric "is an abstraction layer that hides most of the complexities of messaging from the developers") and 20:38-40 ("The Virtual Fabric (fxVF) provides an abstraction layer for application to communicate with each other whether they are in the fxDevice or fxCloud. Various frameworks and services may use the fxVF service.")

C. Software Layers Of Platform Architecture

33. "FIG. 4 is an exemplary view of the *software layers*" or "*software stack*" in the fxOS architecture. *Id.*, 2:3-5, 11:63-64 (emphasis added). An annotated version of Figure 4 is reproduced below. The hardware resources are at the bottom of the stack and include "wireless baseband SoC 402, hardware packet

processor 404, Wide Area Network (WAN) uplink interface 406, cache and storage 408, and multi-core CPU 410." *Id.*, 12:1-4. The device drivers are at the lowest software layer which is positioned closest to the hardware resources. The device drivers function to connect the hardware resources to the operating system kernel, which is positioned above the device drivers in the stack. *Id.*, 11:66-12:1.



34. Above the kernel, there are several tools 416, native daemons 418, native libraries 420 and Hardware Abstraction Layers (HAL) 422. These tools abstract out the hardware dependencies for the upper layers." *Id.*, 12:18-21. Significantly, the "upper layers" must be positioned at a level in the stack above tools 416, native daemons 418, native libraries 420 and Hardware

Abstraction Layers (HAL) 422, otherwise these tools would not be able to abstract hardware dependencies "*for the upper layers.*" The virtual fabric (discussed in Section II(C), *supra*), which hides the complexity of communications between different components of a distributed application and makes them abstract for the developer, is included in native daemons 418. *Id.*, Figure 5 (illustrating "virtual fabric 504e" within "fx-Daemons"). Continuing to move up the stack, the Runtime 424 ("an embedded virtual machine capable of securely isolating and executing the applications") is the next layer, and positioned above that are System Services 426 which "are a set of services always running and available to the developers (e.g., timing and messaging services)." *Id.*, 12:21-26.

35. Finally, the fx-Framework Library 428 is positioned at an *upper layer* of the software stack and consists of several Frameworks which support "*upper layer*" APIs recited in claim 1 of the '823 Patent. As explained in the '823 Patent:

The new frameworks are the library extensions introduced explicitly for the dSDN framework. Each of these frameworks adds a set of methods of (functions) and data structures for the following examples:

Wireless Manager 428a: manipulation, monitoring, and configuration of the wireless interfaces; ...

Big Data Manager 428e: manipulation, processing, and organization of data collected from various elements. This framework could enable close coordination between the fxDevice 302 and the fxCloud 304;

Content Manager 428f: for data sharing (database sharing) between the applications within the fxDevice 302 or the fxCloud 304 and between applications in fxDevice 302 and fxCloud 304....

Policy Manager 428g: responsible for execution and implementation of policies set up by the admin. The application developers could use this framework to perform queries about the permissions, limitations, and rules;

Security Manager 428 h: allows for access to the security protocol libraries and rewriting some of security algorithms such as the man-in-the-middle detection algorithm;

General Framework: general compute and logic building that may be inherited from existing general OS frameworks. *Id.*, 12:27-64.

36. The '823 Patent explains that these frameworks positioned at an upper layer of the software stack can be accessed by applications by way of an API. *Id.*, 12:64-13: 1 ("Using the frameworks, there are potentially at least two types of application types that are possible. First, the System Native Applications 430 that are provided as the initial application load into the platform. These applications could be used by other applications by links or API.")

D. APIs For Communications Between Different Distributed Application Or With Entities Outside Of The dSDN Platform

37. The '823 Patent goes on to disclose the use of custom application programming interfaces (or APIs) that are controlled by the policy and security provisions of the virtual fabric and allow one distributed application to communicate with another distributed application (inter-app communications). As explained in the '823 Patent:

This framework allows the developer to create custom APIs and make it accessible for other applications (in the fxDevice or in the fxCloud).

In turn, the fxCloud could *present* these APIs using e.g. REST technologies (via the fxCloud Northbound Interface) to the developer outside the dSDN system. *Id.*, 28:49-54 (emphasis added).

38. The "fxCloud Northbound Interface" through which the fxCloud presents (or exposes) these custom APIs for communication outside the dSDN system is illustrated in Figure 3 (reproduced below with red coloring added):

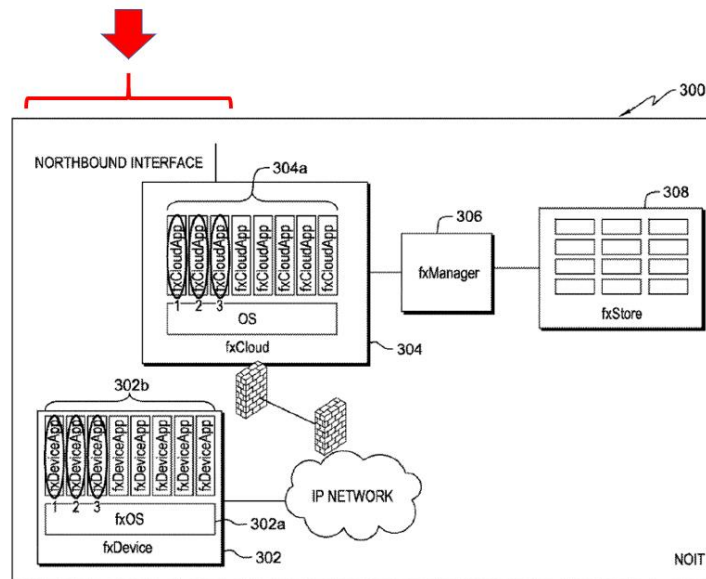


FIG. 3

Because the “fxCloud Northbound Interface” encapsulates the interface of fxDevice 302 and fxCloud 304, it is above the software layers described in connection with Figure 4 of the '823 Patent, and therefore allows programming of both devices.

E. Examples Of Upper Layer APIs Disclosed In The '823 Patent

39. The '823 Patent discloses examples of how the upper layer APIs described above can be used by a service provider to program an *operating*

distributed application from outside the dSDN platform. As explained previously, the wireless manager 428a (shown in Figure 4) provides upper layer APIs for "manipulation, monitoring, and configuration of the wireless interfaces." *Id.*, 12:32-33. In examples of upper layer APIs associated with the wireless manager 428a, the '823 Patent discloses two custom APIs that allow a cellular service provider to program a distributed application that is executing (an operating cellular network), so as to limit access to the cellular network in a case of national security. The '823 Patent describes this example as follows:

In case of national security, the service provider may limit the access to the network so only the law enforcement can use the cellular network. In this case, the admin defines a Target Area (TA) using an fxCloudApp in the fxCloud where the emergency lockdown needs to be applied. The IMSI/MSISDN number of law enforcement mobile devices are sent to the fxDevice in the TA which in turn enforce the policy. The policy may allow the public to just sent SMS while the law enforcement could have full access and priority to the system. *Id.*, column 31, Use #24.

40. In this example, an exemplary distributed application (used for implementing a cellular network) is controlled (or programmed) when there is a national security event in a manner that changes the network access rules from ones which provided members of the public with full access to the system to ones that restrict members of the public to SMS messages while providing law enforcement with full system access. In order to allow the service provider to control (or program) the distributed application in a manner that changes the access rules

(giving priority to law enforcement priority to the network), the '823 Patent discloses exposing the following two custom APIs which can be used to control (or program) the exemplary distributed application (the cellular network):

DataPath {matchTraffic, setPriority}
Security {configureFirewall} *Id.*

41. A POSITA would have understood that these APIs were intended to be accessed by the cellular service provider when the distributed application implementing the cellular service was already running, in order to program the cellular service application to provide law enforcement with priority to the cellular network.

F. Explicit API Frameworks And Platform APIs

42. The '823 Patent introduces "API Frameworks" as library extensions for the distributed Software Defined Networking (dSDN) framework, and upper layer "Platform APIs" that are used to program applications.

- "fx-Framework Library 428 consists of several Frameworks. The new frameworks are the library extensions introduced explicitly for the dSDN framework." ('823 Patent, 12:27-30)

- "The fxOS 302a controls the access of applications to platform APIs." ('823 Patent, 11:53-54)

43. The upper layer APIs abstract complex operations into simple, usable operations. Examples of this follow:

- a. Wireless Framework: Provides functions for "managing and controlling the wireless module," such as ``discoverNeighbors``, ``measureInterference``, and ``setWirelessConfig``. ('823 Patent, Table 8, 34:42-44).
- b. Security Framework: Offers high-level security functions like ``setDOSDetect``, ``configureFirewall``, ``encryptTraffic``, and ``authenticateUser``. ('823 Patent, Table 8, 30:15) Developers can invoke these methods without implementing the underlying security algorithms.
- c. FastPath Framework: Allows developers to "identify and manipulate the data path with a DPI capability" using methods like ``matchTraffic``, ``detectCongestion``, and ``routeTunnel``. ('823 Patent, Table 3, Col 30) This abstracts low-level packet processing and network routing.
- d. Messaging Virtualization: The system creates a "virtual fabric (fxVF)" to abstract messaging complexity for developers."The messaging complexity may be abstract for the developer." ('823 Patent, 11:61-62). This allows applications to communicate without needing to manage network topologies or direct message routing.
- e. Extensible API Framework for Developer Customization: Developers can create custom APIs and expose them, even to external systems using

technologies like REST, indicating a high level of flexibility and interoperability. "This framework allows the developer to create custom APIs and make it accessible for other applications... The fxCloud could present these APIs using e.g. REST technologies (via the fxCloud Northbound Interface) to the developer outside the dSND system." ('823 Patent, 28: 49-56)

44. The '823 Patent's "upper layer" concept is tied to dSDN framework libraries that expose infrastructure programmability primitives (e.g., flow/data-path manipulation and coordination with fxCloud). (Ex. 1001, 12:25–39.)

45. Thus, the '823 Patent provides higher-level APIs that abstract functionalities and services, allowing application developers to command complex system behaviors (e.g., manage network traffic, apply security policies, control wireless devices) with simplified method calls.

46. As discussed more fully below, the intrinsic disclosures show that the claimed "upper layer APIs" are directed to, hardware-abstracted APIs enabling programmability of distributed applications across device and cloud infrastructure—not simply "APIs exist because software is written in Java." (Ex. 1001, 11:54–60; Ex. 1001, 12:20–22; Ex. 1001, 12:25–39.)

G. The Achievements Of The '823 Patent

47. The '823 Patent (also referred to as Taaghol) introduces a groundbreaking Distributed Software Defined Networking (dSDN) architecture designed to revolutionize how resources are managed and operated. At its core, Taaghol envisions an "infrastructure-centric" paradigm where the network itself—from devices to cloud resources—becomes a highly programmable and flexible entity. This system provides a unified abstraction layer, allowing for the comprehensive lifecycle management of services and infrastructure applications, independent of specific hardware vendors. This stands in stark contrast to Vasell, which presents a "service gateway system" (e-box) primarily focused on delivering specific, localized consumer services, treating the underlying hardware and software as distinct silos.

48. Taaghol's primary objective is to enable secure and flexible programmability across the entire network, simplifying its deployment and management from conception to decommissioning. This ambition goes beyond merely installing software on devices; it aims to imbue the network infrastructure with inherent programmability. The patent explicitly states this end-to-end scope:

"The Distributed Software Defined Network (dSDN) disclosed herein is an end-to-end architecture that enables secure and flexible programmability across a network with full lifecycle management of services and infrastructure applications (fxDeviceApp). The dSDN also harmonizes application deployment across the network independent of the hardware vendor." (U.S. Patent 11,695,823 B1, Page 8, Column 2, lines 58-63).

49. This clearly defines Taaghol's focus on the entire network as a programmable domain, allowing deep control over its functions such as routing, switching, security, and deep packet inspection. In contrast, Vasell's "e-box" is designed to perform specific user-facing tasks like remote meter reading or home automation, operating as a distinct service delivery point rather than as a programmable segment of a larger, unified network infrastructure. Vasell's "programmability" focuses on downloading small Java applets ("boxlets") that perform these specific user-facing tasks, which is a localized application management function, not unified control over network infrastructure.

50. A cornerstone of Taaghol's innovation is its insistence on "unified capabilities" and "hardware independence," which collectively create a novel programming environment. Taaghol mandates a consolidated programming environment where both network device applications (fxDeviceApp) and cloud applications (fxCloudApp) can be controlled through a single, consistent paradigm, abstracted from the underlying hardware. This addresses the fundamental inflexibility of traditional proprietary network solutions.

51. As described in the excerpt above, the patent emphasizes the harmonization of application deployment across the network regardless of the underlying hardware vendor. Taaghol's architecture decouples the control logic from the underlying hardware. This allows a single set of APIs to configure

disparate network devices and cloud resources as if they were a unified, programmable entity, removing the "silos" that characterize other systems. It describes a sophisticated, abstract layer within the dSDN architecture that merges the control planes of the device and the cloud. This unified view simplifies deployment and drastically reduces operational overhead by allowing network behavior (e.g., routing logic, security policies) to be expressed at a high level of abstraction, unlike Vasell's "integrated" physical bundling of components. Vasell's "integration" refers to the physical bundling of components within a device, like a transponder integrated with a GPS receiver, rather than a logical unification of control planes across a distributed, hardware-independent network.

52. Taaghol's dSDN architecture relies on a rich set of "upper-layer application programming interfaces (APIs)" to achieve programmatic control over network functions. These APIs allow developers to interact with the running logic of network elements, configure parameters, and define behaviors, such as routing, switching, security, and deep packet inspection (DPI), all through a unified abstraction layer. Taaghol's vision for these APIs is to abstract hardware dependencies, enabling higher-level programming:

"These tools abstract out the hardware dependencies for the upper layers and programmers." (U.S. Patent 11,695,823 B1, 12:20-22)

53. This means developers using Taaghol's system don't need to write code specific to a particular vendor's hardware. Instead, they interact with abstract

functions that operate consistently across the distributed infrastructure. The patent details several specialized frameworks (Wireless, FastPath, Security, Management) providing methods for deep infrastructure control, such as ``setWirelessConfig``, ``configureFirewall``, ``matchTraffic``, and ``upgradeFirmware``. Furthermore, Taaghol offers an "Extensible API Framework" that empowers developers to create custom APIs accessible across the entire dSDN:

"This framework allows the developer to create custom APIs and make it accessible for other applications (in the fxDevice or in the fxCloud). In turn, the fxCloud could present these APIs using e.g. REST technologies (via the fxCloud Northbound Interface) to the developer outside the dSND system." (U.S. Patent 11,695,823 B1, 28:50-58)

54. This demonstrates a dynamic system where custom, high-level APIs can be created and made available across both network devices and the cloud. This capability empowers runtime programmatic control over distributed functionality via a unified, accessible interface, allowing applications to "execute a particular procedure" (U.S. Patent 11,695,823 B1, 29:5-6) directly impacting network behavior at runtime.

55. The architectural components of Taaghol's dSDN work in concert to deliver this infrastructure-centric vision, featuring flexible network devices (fxDevice), a flexible cloud platform (fxCloud), an application management portal (fxManager), and an infrastructure application marketplace (fxStore). These elements form a "single virtual view of the system" (U.S. Patent 11,695,823 B1,

13: 63-64), which is foundational for unified control. Each fxDevice runs a sandboxing operating system (fxOS) that securely isolates applications and allows efficient execution, including FastPath processing for data plane packets and Deep Packet Inspection (DPI).

56. The dSDN supports sophisticated features like "Cloud Breathing" (automatic expansion/reduction of cloud resources based on load), Distributed Resource Services (dRS) for managing platform resources and inter-app APIs, and Distributed Content Service (dCS) for seamless data sharing. These capabilities collectively demonstrate a deep integration and programmatic control over the entire network ecosystem. The system also integrates security as a fundamental component, enabling "uniform policy (including security) distribution and execution across dSDN" components.

"The dSDN system 300 consists of a flexible network device (fxDevice) 302, a flexible cloud platform (fxCloud) 304, an application management portal (fxManager) 306, and an infrastructure application market place (fxStore) 308." (U.S. Patent 11,695,823 B1, 10:9-13)

57. In summary, U.S. Patent 11,695,823 by Taaghoh presents a truly infrastructure-centric, unified, and hardware-independent Distributed Software Defined Networking architecture. It aims to transform the entire network into a programmable fabric through "unified capabilities" and upper-layer APIs that abstract away hardware complexities, enabling deep programmatic control over

network functions and distributed applications as a whole. This allows network behavior to be defined and managed at a high level of abstraction, harmonizing heterogeneous elements into a single virtual view.

VI. Claim 1 Of The '823 Patent

58. The '823 Patent includes two independent claims, namely, claims 1 and 19. Claim 1 is representative:

- [1.0] 1. A system comprising:
- [1.1] a programmable network device adapted to host a plurality of network device applications;
- [1.2] a programmable cloud device adapted to host a plurality of cloud applications, [1.3] wherein the plurality of network device applications and the plurality of cloud applications are in secure communication with each other to form distributed applications; and
- [1.4] wherein the plurality of network device applications and plurality of cloud applications [device] form unified capabilities enabling a plurality of upper layer application programming interfaces (APIs) to program the plurality of network device applications and plurality of cloud applications independent of network device hardware and cloud device hardware.

59. A POSITA, reading claim 1 in the context of the specification of the '823 Patent would have understood that claim 1 has the meaning set forth below:

- a. First, Element [1.2] makes clear that "the plurality of first network applications" and "the plurality of second network applications" form "distributed applications." Thus, a POSITA would have understood that, as referenced throughout the claim, "the

plurality of first network applications" and "the plurality of second network applications" are not disparate, unconnected applications but rather are parts of "distributed applications." A POSITA would have understood that the claim language defining distributed applications corresponds, for example, to the disclosure in the specification that explains that a "distributed application" is formed from the combination of an fxDeviceApp 302 together with its "sister app" in the backend cloud infrastructure and referred to as fxCloudApp 304a. Ex. 1001, 10:25-31 ("fxDeviceApp 302b may have a sister app in the backend cloud infrastructure (i.e., flexible cloud platform 304) referenced here as fxCloudApp 304a. The fxCloudApp 304a in the cloud is paired with its fxDeviceApp 302b in the fxDevice 302. The fxCloudApp 304a and the fxDevice App 302b collectively form a distributed application (dApp or fxApp).")

b. Second, Element [1.4] recites "a plurality of upper layer application programming interfaces (APIs)." A POSITA would have understood that, in the context of the specification of the '823 Patent, upper layer APIs are not APIs such as device drivers positioned at a low layer in the software stack (i.e., close to the hardware). Rather, examples of upper layer APIs in the specification include the APIs in

fx-Framework Library 428, which are positioned at the highest software layer in the stack, just under the applications themselves. See, *id.*, at Figure 4 (illustrating fx-Framework Library 428 immediately below fxApps 308b).

c. Third, Element [1.4] of claim 1 recites that the "plurality of upper layer application programming interfaces (APIs)" function to "program the plurality of network device applications and plurality of cloud applications" (i.e., the distributed applications) in a manner that is "independent of network device hardware and cloud device hardware." In the example shown in Figure 4 of the specification, the '823 Patent provides the claimed independence from the network device hardware and cloud device hardware at the upper layer APIs by using, for example, abstraction layers below the upper layer APIs such as tools 416, native daemons 418 (which include a virtual fabric for hiding the complexities of messages), native libraries 420 and Hardware Abstraction Layers (HAL) 422 that "abstract out the hardware dependencies for the upper layers." *Id.*, 12:18-21 and 13:36-37. A POSITA would have understood that interfaces positioned close to the hardware, e.g., device drivers, are not "upper layer" application programming interfaces as used in the '823 Patent.

d. Fourth, Element [1.4] of claim 1 recites that the "plurality of upper layer application programming interfaces (APIs)" are enabled "by unified capabilities" to "program the plurality of network device applications and plurality of cloud applications" which were previously defined in Element [1.2] as forming the distributed applications. A POSITA would have understood that the "unified capabilities" enabled by the upper layer APIs provide the user with the capability to program the distributed application as whole or holistically, without needing to have an understanding of which portion of the distributed application comprises the network application component and which aspect of the distributed application comprise the cloud application component. The specification discloses an example of the "unified capabilities" enabled by the "upper layer APIs" in connection with the programming of a cellular network (implemented as a distributed application) upon the occurrence of a national security event, discussed in Section II(G), *supra*. There, the '823 Patent discloses custom APIs used to control (or program) the distributed application, which are exposed as follows:

DataPath {matchTraffic, setPriority}
Security {configureFirewall}

Using such upper layer APIs, the user can program the Datapath or Security of the cellular network holistically (or, as a whole) without needing to have an understanding of which portion of the distributed application comprises the network application component and which aspect of the distributed application comprise the cloud application component. Thus, a POSITA would have understood that, in the context of the '823 Patent, the "unified capabilities" enabled by the "upper layer APIs" are referring to providing the user with the capability to program a distributed application holisitcally (or, as a whole) without necessarily needing to have an understanding of which portion of the distributed application maps to the network application components and which aspect of the distributed application maps to the cloud application.

e. Fifth, Element [1.3] of claim 1 specifies that "the plurality of network device applications and the plurality of cloud applications *are in secure communication with each other* to form distributed applications." Here, the word "are" is used as a *present tense* verb, which means that in the claimed system, the plurality of network device applications and the plurality of cloud device applications must *be* securely communicating (*present tense*) with each other. A

POSITA would have understood that this claim language necessarily requires that the plurality of network device applications and the plurality of network device applications forming the distributed applications be in a state where they are in fact operating (or running) -- otherwise, such applications could not be in secure communication with each other. A POSITA would have understood that, by referring back to "the plurality of network device and plurality of cloud applications," the final wherein clause (Element [1.4]) makes clear that the unified capabilities enable a plurality of upper layer APIs "to program" applications that are in secure communication with each other (and therefore in an operational state).

f. In summary, as used in Element [1.4], an “upper layer application programming interface” that is enabled “by unified capabilities” to “program the plurality of network device applications and plurality of cloud applications independent of network device hardware and cloud device hardware” means an upper layer interface for holistic runtime¹ programming of a distributed application as a whole, and where the unified interface abstracts away the

¹ The use of "runtime" is meant to signify that the distributed applications are in a state where they are operating (or running).

heterogeneous characteristics of both the network device hardware and the cloud device hardware.

60. The element in claim 1 that recites "wherein the plurality of network device applications and plurality of cloud applications form unified capabilities enabling a plurality of upper layer application programming interfaces (APIs) to program the plurality of network device applications and plurality of cloud applications independent of network device hardware and cloud device hardware," establishes a stringent requirement for "unified capabilities" that go beyond mere connectivity. This mandate describes a sophisticated, abstract layer within a distributed software-defined networking (dSDN) architecture, aiming to create a consolidated programming environment for both network device applications and cloud applications through a single, consistent paradigm. This effectively merges the control planes of the device and the cloud, normalizing heterogeneous network elements into a unified view that is independent of specific hardware. This design enables a high level of abstraction for network programmability, providing a cohesive environment for managing diverse applications and infrastructure elements.

61. This claim element highlights a foundational principle: a single, coherent set of "unified capabilities" that allows for programmatic control over both network and cloud applications without needing to account for the underlying

hardware. This approach is designed to manage heterogeneous elements by normalizing them into a unified view.

62. Figure 3 from '823 patent shows the holistic interaction between network devices and cloud resources. Figure 3 depicts the functional system architecture where the "fxDevice" (network device) and "fxCloud" (cloud device) are merged into a single, unified control plane.

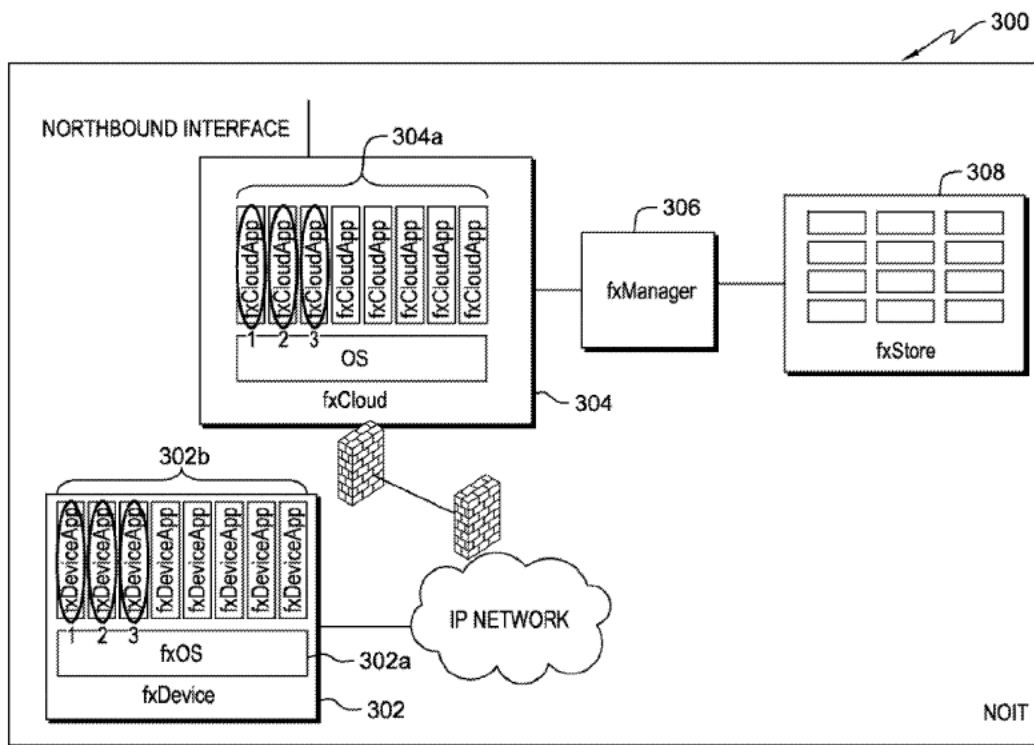


FIG. 3

63. The figure shows the fxDevice 302 (the edge hardware) and the fxCloud 304 (the cloud infrastructure) communicating via a secure control path. The '823 Patent further clarifies that the dSDN "harmonizes application deployment across the network independent of the hardware vendor" ('823 Patent,

2:36-38), signifying a logical unification of control rather than just physical connectivity. This creates a "single virtual view of the system" through the fxCloud ('823 Patent, 13:61-63), which interacts with and manages applications, firmware, and the operating system on the fxDevice. This is further clarified in the following excerpt:

“In the dSDN system, fxDeviceApp 302*b* may have a sister app in the backend cloud infrastructure (i.e., flexible cloud platform 304) referenced here as fxCloudApp 304*a*. The fxCloudApp 304*a* in the cloud is paired with its fxDeviceApp 302*b* in the fxDevice 302. The fxCloudApp 304*a* and the fxDeviceApp 302*b* collectively form a distributed application (dApp or fxApp). In this description, when fxDeviceApp is referenced it shall mean any application that may have software components in the fxDevice, fxCloud, or both. In general, an fxApp package may include the following software components:

fxDeviceApp binary;
fxCloudApp binary;
Manifest files; and
Signatures.

It is important to note that the fxDevice 302*b* and the fxCloud 304*a* could use any protocol to communicate with each other and there is no need for standardizing this communication allowing for ultimate freedom for the developers. This allows for the system to operate in a loosely coupled autonomous fashion allowing for asynchronous communication in a distributed fashion.” ('823 patent, 10:25-46)

64. Effectively 823 patent defines a "distributed application" (fxApp) as a collective entity formed by pairing software across the device and cloud. By treating device and cloud software as a unified whole, the system enables for developers a truly holistic runtime programming model that spans the entire

distributed fabric. In my opinion, using the syntax portability (allowing code to run on different hardware) of a high level language like Java as an API does not meet the "upper layer" requirements of the claim language.

VII. STATE OF THE ART - JAVA DEVELOPMENT ENVIRONMENT

65. Before getting into the details of the Grounds advanced in the Petition, Patent Owner provides the following background on the state of the art at the time of the '823 Patent.² The standard Java Platform included “classes” and related “APIs.” A Java class defined the data (variables, also known as fields or attributes) and methods (functions that operate on that data) for objects in a class. A class did not represent a physical entity itself but rather provided the specifications for

² The standard Java development platform was discussed in numerous references considered by the Examiner and listed on the face of the '823 Patent. *See, e.g.*, U.S. Patent Publ. No. 2004/0107417 [Ex. 2007], [0057] (referring to “a java virtual machine”) and [0071] (referring to “Sun’s Java platform standard”); U.S. Patent No. 8,495,611 [Ex. 2008], 10:16-17 (referring to “Java Development Kit (“JDK”) software”); U.S. Patent Publ. No. 2008/0209491 [Ex. 2009], [0085] (referring “object-oriented environments such as “Java”); U.S. Patent No. 7,630,706 [Ex. 2010], 1:46-56 (referring to “a Java virtual machine” and “Java applications”); U.S. Patent Publ. No. 2011/0265077 [Ex. 2011], [0020] referring to a “Java web application” and “Java classes”); U.S. Patent Publ. No. 2012/0066665 [Ex. 2012], [0038] (discussing applications based on “Java”); U.S. Patent Publ. No. 2012/0266156 [Ex. 2013], [0051] (discussing a “Java Runtime Environment” and a “Java virtual machine (JVM)”); U.S. Patent Publ. No. 2013/0007254 [Ex. 2014], [0022] (referring to “Java Virtual Machines”); U.S. Patent Publ. No. 2013/0263104 [Ex. 2015] (referring to “an object oriented programming language such as Java”); and U.S. Patent Publ. No. 2014/0052976 [Ex. 2016], [0145] (disclosing Java software implementation).

creating instances of that entity, which were called objects. For example, a "Car" class would define the properties (color, make, model) and actions (accelerate, brake) that all car objects would share.

66. At the time of the '823 Patent, the JAVA Platform included a standard set of class libraries and APIs. Ex. 2006, 7 (describing Java "Base Platform"). A class library was a collection of pre-written Java classes and interfaces that provided specific functionalities that could be imported and utilized in Java applications, saving developers from writing code for common functionalities from scratch. Ex. 2006, 8 (explaining that Java's set of APIs allowed developers to build "applications from shared, reusable objects" which functioned to "reduce cost by allowing developers to concentrate on creating only what is novel"). The functionality implemented using the standard Java APIs available at the time provided "very basic language, utility, I/O, network, GUI, and applet services," and was therefore implemented at a low layer close to the hardware. Ex. 2006, 14.

67. At the time of the '823 Patent, developers used the Java language to write standalone applications. Ex. 2005, 24. In order to execute a Java program, Java source code (which included references to APIs associated with the standard class library) was compiled into the form of bytecodes. *Id.*; Ex. 2006 ("Programs written in the Java Language and then compiled will run on the Java Platform"). Prior to running the Java application in the Java virtual machine, the Java class

libraries associated with the APIs referenced in the source code (and thus needed to execute the bytecode) were loaded by a class loader and provided along with the bytecode to a Java Interpreter in the Java Virtual Machine, which then ran the application. *Id.*, 24-25; Ex. 2006, 24 (“Once in the Virtual Machine, the bytecodes are interpreted by the Interpreter ... Any classes from the Java Class Libraries are dynamically loaded as needed”). A graphical representation of these operations is depicted below:

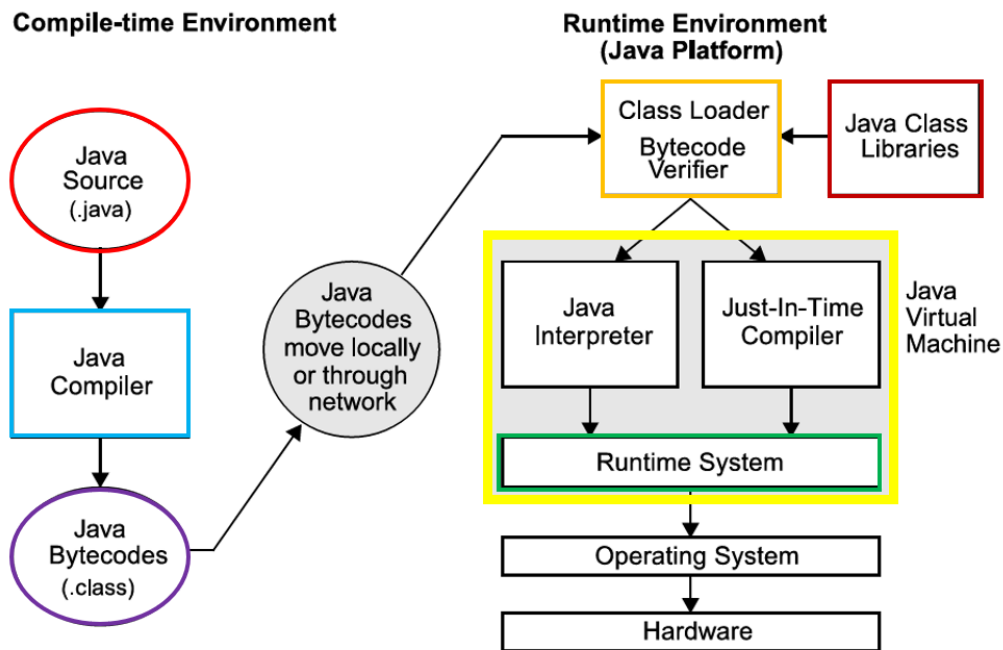


Figure 3. Source code is compiled to bytecodes, which are executed at runtime.

68. Referring to the diagram above, a developer initially wrote Java Language source code (.java files) (red oval) using, *inter alia*, APIs associated with pre-written Java classes stored in Java Class Libraries. The developer next used a Java Compiler (blue box) to compile the source code into bytecodes (.class files)

(purple box). Ex. 2006, 24. The developer's code needs to meet the requirements of the API at compile time. To run the application, the bytecodes were loaded into memory and verified for security (orange box) before they entered a Java Virtual Machine (yellow box). *Id.* In addition, a class loader (orange box) loaded the Java class libraries (brown box) associated with APIs referenced in the source code. *Id.* Once in the Java Virtual Machine, the bytecodes were interpreted by either an Interpreter or a just-in-time (JIT) code generator inside the Java Virtual Machine. *Id.* After being processed by either the Java Interpreter or the JIT code generator, the application was run in the Runtime System (green box) within the Java Virtual Machine.

69. There is a critical distinction between the Java APIs used to write source code (as described above) and the APIs recited in the final wherein clause of the '823 Patent. Specifically, the Java APIs were used to write the original source code (red box) underlying the application itself, rather than to program the application *after* it was operational in the Runtime System (green box).

VIII. Prior Art Relied On In Petition

A. Vasell

70. Patent 6,496,575 B1 by Vasell et al. outlines an "Application and Communication Platform for Connectivity Based Services," centered around a "Service Gateway System" or "e-box." This system is fundamentally "service-

centric," focusing on the deployment and management of discrete applications that perform specific user-facing tasks within residential or localized environments. Its core objective is to provide an open platform for various service providers to load software onto a dedicated server, which then interacts with remote devices to implement subscribed services.

71. Vasell's "service platform server" (the e-box) is envisioned as an edge server installed at the user's location, directly connected to a local area network (LAN). Its role is to interact with a plurality of remote devices such as thermostats, washing machines, coffee makers, and telephones, to implement services like environmental control, laundry management, or telephony. The system facilitates the transport of these applications and enables the gateway to communicate with local peripherals through specific hardware drivers. This localized scope—interacting with devices predominantly at the user's premises—is a defining characteristic of Vasell:

"The major components of an exemplary Service gateway system embodiment for residential use are depicted FIG. 2. As shown, a local area network 10 may be provided in a residence for communicating with and controlling a plurality of sensors, appliances and instruments. Those depicted include a thermostat 12, a clothes washing/drying appliance 14, a coffee maker 16, and a telephone 18." (Vasell, 5:36-42)

72. This description vividly portrays the service-centric nature, focusing on specific consumer devices and home automation within a residential

context. The gateway acts as a hub to enable and manage these localized services for the homeowner. In stark contrast, Taaghol's Distributed Software Defined Networking (dSDN) architecture operates on a much broader canvas, aiming to control the entire network infrastructure across diverse network devices and cloud resources, treating them as a single, programmable entity that is independent of specific hardware.

73. A key architectural element of this system involves "cells" (Vasell, Figure 6, 90-92), which differ from conventional software processes by not having parent/child relationships and being unable to start or stop each other. The number and identities of cells are configured via a cell table (Vasell, Figure 6, 94), providing a more controllable and static environment. Each cell has a unique name, can be designated as permanent or transient, and is defined by a quota of resources, including CPU, memory, and persistent storage. In this architecture, nearly every object belongs to a specific cell, ensuring a structured and managed environment for service applications (Vasell, Figure 6, 70-72).

74. Inter-cell communication are primarily managed by "gates" (Vasell, Figure 6, 80-86) and class loaders (Vasell, Figure 6, 97). Gates serve as controlled interfaces (Vasell, Figure 6, 80-86), facilitating communication between cells (Vasell, Figure 6, 90-92), the main services layer (Vasell, Figure 6, 100), and the system services layer (Vasell, Figure 6, 110), effectively preventing direct object

references between cells. Each gate is specific to a pair of cells and operates locally within environments like a Java Virtual Machine (JVM). Class loaders (Vasell, Figure 6, 97) are utilized to implement cells, providing encapsulation, protection, and access restriction for service applications.

75. The "programmability" described in Vasell is primarily focused on Application Lifecycle Management (ALM). This involves the ability to download small Java applets ("boxlets") onto the service platform server, along with instructions for their installation, execution, and termination. For instance, a network operator can download a portion of a service application (e.g., environmental control software) to the user's gateway, which then runs this software to monitor and control a thermostat.

"The management system service 76 provides an external interface through which the service applications 70-72 may be downloaded, installed, removed, executed, and controlled." (Ex. 1004, 22:32-38)

76. This process is about getting pre-defined software binaries onto a device and managing their state. This is distinctly different from Taaghol's concept of programmability, which centers on programmatic control over the network infrastructure itself—including routing, switching, security, and deep packet inspection—via a unified abstraction layer, allowing for dynamic definition of network behavior at runtime, independent of the underlying hardware. Taaghol's

"upper-layer APIs" are designed to program the logic of network functions, not just manage the state of installed applications.

77. A critical aspect of Vasell's architecture is its inherent "hardware dependency," which directly contradicts Taaghol's pursuit of hardware-independent control. Vasell's "software interfaces" are often closely tied to lower-level, hardware-specific communication protocols and device drivers. Vasell explicitly states that its "local network services 78" provide "proper interfaces and drivers which allow the service applications 70-72 to communicate with the devices 12-18 on the local network 10." It further elaborates on this dependency:

"This service may wrap, for instance, serial port drivers (e.g. for the X10 protocol) or Ethernet drivers (e.g. for the CEBus protocol)." (Vasell, 22:46-49)

78. This indicates that Vasell's applications interact directly with, or through wrappers for, specific physical layer protocols and drivers (like X10 for power line communication or CEBus for home automation). Such interfaces are intrinsically linked to the physical characteristics of the local devices and networks. This direct reliance on hardware-specific drivers means that programming for new or different hardware would necessitate modifying or creating new drivers, potentially limiting hardware independence. Taaghol, conversely, explicitly aims to abstract away these hardware dependencies, stating that its tools "abstract out the hardware dependencies for the upper layers and programmers" (Taaghol,

Column 2, lines 6-7), enabling a single API to configure diverse network and cloud resources irrespective of their underlying hardware.

79. The "interfaces" in Vasell are primarily geared towards application lifecycle management and localized communication, not "upper-layer APIs" that offer a unified programming model across disparate domains. While Vasell mentions "Java application program interfaces (API) that comply with mainstream Java development" (Ex. 1016, at 4), these are internal implementation details or local communication channels used by applications to access resources or interact with other local components. They are not exposed externally as a high-level, hardware-independent abstraction layer for programming network and cloud functions. These APIs enable interaction within the gateway's defined environment, allowing applications to utilize local services and resources, but they do not abstract away underlying hardware specificities to program heterogeneous network device and cloud applications as a cohesive, distributed entity.

80. Furthermore, Vasell's system embodies a "siloes service delivery" model. Each service, though potentially sharing infrastructure, operates largely as a distinct, vertical slice of functionality. A utility company might send an application to the gateway to read a meter, while another application manages home heating. While the system allows multiple service applications to "simultaneously operate without detrimentally affecting each other" (Vasell, 4:59-

61), this refers to resource isolation at the execution environment level (e.g., using Java Virtual Machine cells), not a unified, holistic control over the network fabric itself. The service gateway runs its specific application, and the network operator server runs its management application; they interact via standard network protocols, but they do not share a unified control plane that merges device and server functionalities into a single, programmable domain. This is fundamentally different from Taaghol's vision of a unified network system that presents a single virtual view of the network for seamless control and management of distributed applications, independent of specific vendors.

81. Vasell allows the services to communicate with each other (Vasell, 13:2-12)

“For example, it may be the case that the service application **70** controls the audio system of a business office, and the service application **71**, having as one of its sensors a noise 5 detection unit, controls the security system of the same business office. In this example, it is desirable for the two service applications **70** and **71** to communicate with each other in order to avoid false alarms due to the audio system under the control of service application **70** being detected by 10 the noise detection unit under the control of alarm system service application **71**. “

82. Thus, Vasell teaches how to manage localized service interactions, like applications 70 and 71 coordinating, through internal gates and hardware-specific drivers rather than a system-wide abstraction. This approach lacks the comprehensive hardware independence and generalized API control needed to

program diverse network and cloud resources as a unified entity. Therefore, it focuses on specific application-level coordination within a gateway, not unified infrastructure programmability.

83. Vasell's use of the term "integrated" refers predominantly to the "physical bundling" and internal coherence of components within its architecture, rather than the logical unification of control planes or hardware-independent abstraction across a distributed network. For example, it describes how data processing and communication devices become "further integrated" (Vasell, 1:40:40), or how a particular service can be "integrated" into the gateway. This form of integration is about assembling different parts—radios, sensors, processors—within a specific device to form a functional unit, or streamlining the internal management of services within the gateway system. It focuses on how components work together within a defined boundary, rather than creating a comprehensive, software-defined programming interface that spans across a distributed network and merges control planes for unified control.

"As devices which process and manipulate data become further integrated with devices which communicate data, the potential to use such integrated technologies expands their applicability." (Vasell, 1:39-42)

84. This statement highlights physical consolidation within a device, not a logical unification across disparate network elements. In stark contrast, Taaghoh's "unified capabilities" explicitly aim to merge control planes for network devices

and cloud applications, abstracting hardware specifics to provide a single programmable domain.

85. In essence, U.S. Patent 6,496,575 by Vasell describes a system where components are physically integrated to deliver specific services, and interfaces are employed to manage software installation or facilitate particular communication links, often with significant hardware dependencies. It focuses on getting software onto a remote "e-box" and managing its state (Application Lifecycle Management) to perform specific user-facing tasks.

B. Alves

86. Alves (as provided by excerpts from the book in Exhibit 1008) outlines how OSGi addresses Java's inherent limitations in modularity by providing a framework for building and managing applications as interconnected modules. The excerpts provided are from chapter 1, chapter 4 and chapter 10. A portion of the summary provided by Alves (at the end of the chapter) is as follows:

“Development platforms consist of an application framework and a supporting runtime environment. The Java platform, with the JDK and JRE, is one example of a development platform. Enterprise platforms, such as Java Enterprise Edition (JEE), add enterprise features to the platform.

The OSGi Service Platform provides a dynamic Java module system for Java. It allows Java code to be modularized and to be managed as services. The OSGi Service Platform consists of the OSGi framework and the OSGi services. The OSGi Service Platform Enterprise Specification (Enterprise OSGi) was created to support enterprise use

cases. It defines a collection of OSGi services that can be used together for enterprise features.

OSGi provides the means to achieve modularization, which helps decrease and manage the complexity of large systems. OSGi provides a native extensibility mechanism through the use of services.” (Ex 1008, Chapter 1, pg. 16 (as per the book), Section 1.7)

87. A portion of the summary provided by Alves (at the end of the chapter) of Chapter 4 is as follows:

“The OSGi framework provides several options for minimizing bundle dependencies. You can restrict a resource to be within a private package and thus not visible to any other bundle. You can also choose to export only a subset of the classes residing in a package, thus reducing the number of classes that make up the exported package contract. Finally, you learned that you should avoid creating bundle dependencies using the Bundle-Required header, because it can cause split packages and cause other problems.

We looked into three OSGi features that facilitate the creation of generic bundles. Dynamic imports provide a way out of the explicit import and export package contract, because they allow classes to be searched as needed. Optional packages are regular import packages that aren’t considered an error if they’re not resolved. And finally, fragment bundles are degenerated bundles that attach and merge with a host bundle, possibly extending its class path.” (Ex 1008, pg 130 (from the book), Section 4.8)

C. Hall

88. Hall (as provided by excerpts from the book in Exhibit 1009) mainly outlines OSGi bundles and how to deploy them (pg. 179, 283, 331-334) and security issues (Chapter 14). The content delves into practical aspects like managing memory leaks in OSGi environments, utilizing deployment packages for

structured bundle distribution and updates, and comprehensively detailing the security model, including Java's foundational security concepts, OSGi-specific permissions, conditional permission management, and the use of digitally signed bundles for policy. Hall mentions

“The deployment package design has a few other desirable characteristics. First, the deployment package puts metadata in its manifest, similar to bundles, which allows you to turn it into a named and versioned set of resources. Second, by taking advantage of the fact that JAR files can be signed, you can use signed JAR files to make your deployment packages tamperproof. “(Exhibit 1009, pg. 332)

and

“A deployment package is defined as a set of resources that must be managed as a unit. The resources in a deployment package are assumed to be tightly coupled, such as a bundle and its configuration data. As a consequence, a resource can belong to only one deployment package; otherwise, for example, you could run into situations where you had two different, conflicting configurations for the same bundle” (Exhibit 1009, pg. 333).

and

“In this chapter, we discussed how to manage your OSGi-based applications. We covered the following issues:

- One of the key management tasks is deploying bundles to the OSGi framework. You can use multiple techniques to do so, including rolling your own approach or using technologies like OBR and Deployment Admin.
- OBR focuses on discovering and deploying bundles and the transitive closure of their dependencies, whereas Deployment Admin focuses on defining and deploying sets of bundles and needed resources.
- You can use the Start Level Service to control the relative activation order of your deployed bundles, which may be

needed in a few situations like creating splash screens and different execution modes.

D. Rellermeyer

89. The Rellermeyer paper proposes a modular approach to address dependability challenges inherent in cloud services, highlighting that traditional replication methods based on JAVA often fall short due to difficulties in isolating application state for migration and recovery, as well as maintaining client transparency across replicated instances [Exhibit 1011, Page 1, Abstract, and Page 1, Section I].

90. This core methodology involves extending the OSGi standard with a dependability service called CLOUDDEP [Exhibit 1011, Page 2, Section IV]. CLOUDDEP transforms OSGi services into elastic units that can be migrated and replicated across multiple cloud nodes to increase availability and reliability [Exhibit 1011, Page 2, Section IV]. It offers three key dependability properties: re-deployment, which facilitates application migration to new physical machines or data centers upon platform failure; clustering, which eliminates single points of failure through functional unit replication; and elasticity and load-balancing, which distribute workload across flexible resource pools to prevent overloading single instances [Exhibit 1011, Page 2, Section IV]. CLOUDDEP functions as a global entity, managing configurations and topologies on each node and monitoring service invocations to detect failures [Exhibit 1011, Page 2, Section IV].

91. The Rellermeyer paper focuses on improving dependability and elasticity within cloud services primarily through a modular software approach and the CLOUDDEP service. It discusses software modularity and component coupling, particularly within the context of Java middleware and the OSGi framework, to achieve dependability and elasticity [Exhibit 1011, Page 1, Section I and Section II]. However, this discussion primarily centers on software components and their interactions within a potentially distributed environment.. It does not lay out a detailed, system-level heterogeneous architecture encompassing diverse hardware and software components interacting across a network.

IX. Overview Of Distinctions Between the '823 Patent And Vasell

92. Taaghol's architecture strategically employs upper-level interfaces to abstract the intricate details of hardware and network infrastructure, enabling developers to program and orchestrate distributed applications without requiring deep, low-level operational knowledge. This multi-layered design, visually represented in Figure 4 of the '823 Patent (described above) where interfaces are positioned far from the hardware, deliberately decouples application logic from the underlying complexities. This is achieved through mechanisms like the fx-HAL for hardware abstraction, the fx-Runtime for secure execution environments, and the fxCloud/fxOS creating a fxVF to abstract network messaging, ultimately simplifying developer interaction and promoting a hardware-agnostic approach.

93. This comprehensive abstraction disclosed by Taaghol empowers developers to focus on high-level application logic and innovation, rather than being bogged down by hardware specifics. Taaghol provides unified capabilities through APIs and frameworks such as the Distributed Cloud Storage (dCS) and Distributed Notification Service (dNS), facilitating the creation of sophisticated distributed services that can operate seamlessly across diverse device and cloud environments. This approach fosters innovation by streamlining development cycles, offering an "end-to-end programming platform."

94. By contrast, Vasell's management system service 76 features an external interface primarily designed for the fundamental lifecycle management of individual "service applications" (70-72). A POSITA would have understood that this interface operates at a low layer to administer functions such as downloading, installing, removing, executing, and controlling these software modules. Positioned within the "system services layer" 110, this lower-layer management API focuses on the direct deployment and foundational operational control of discrete software components, ensuring their proper execution and basic runtime behavior. It manages "boxlets" (64-69) within "cells" (90-92) to control resource allocation and enhance security, akin to system-level tasks rather than complex application programming.

95. Vasell's management system services 76 is a low level interface explicitly intended for network operators, who centrally manage the maintenance, operation, and support of the service platform. It offers an operator-centric control model, providing mechanisms for managing the execution environment of pre-defined services. Crucially, it does not support dynamic, high-level programming of a distributed application. Its scope is restricted to foundational (i.e., low layer) system administration, contrasting sharply with higher-level interfaces that abstract infrastructure complexities to enable broad distributed application development and orchestration.

96. The Petition fails to explain how Vasell discloses the "upper layer APIs" required by the '823 Patent. That is because Vasell's teachings are limited to functionality implemented through lower layers, such as localized management and administrative lifecycle tasks. The Petition relies on numerous passages in Vasell that describe high-level architectural benefits, "open platform" design principles, or low-level internal component interfaces—such as Java class libraries and "gate" managers—rather than a hardware-independent programming interface for distributed applications. Unlike Taaghol, which provides high-level frameworks for runtime infrastructure control (e.g., wireless, security, and data-path frameworks) that abstract hardware complexities associated with distributed applications into simplified method calls, Vasell's APIs are restricted to the local

gateway environment. They function as administrative "loaders" to install or execute software binaries rather than tools for defining the runtime behavior of a cohesive, distributed entity spanning both network devices and the cloud.

97. The '823 Patent discloses "unified capabilities" that merge device and cloud control planes into a single, hardware-agnostic paradigm for programming distributed applications—a feature entirely absent in Vasell. Vasell's concept of "integration" refers merely to physical bundling and operational coherence within a single gateway device ("e-box"), relying on conventional network protocols for external communication rather than a consolidated software-defined architecture. Vasell remains tightly coupled to underlying physical hardware, requiring device-specific drivers for different protocols, whereas Taaghol's unified capabilities provide a single virtual view that "harmonizes" deployment and control across disparate vendor hardware. The above limitations of Vasell are described in detail below.

X. Distinctions Between "Upper Layer API" As Set Forth In Claim 1 Of the '823 Patent And Vasell

98. Element [1.4] of Claim 1 of the '823 Patent recites in pertinent part:

wherein the plurality of network device applications and plurality of cloud applications [device] form *unified capabilities enabling a plurality of upper layer application programming interfaces (APIs) to program the plurality of network device applications and plurality of cloud applications* independent of network device hardware and cloud device hardware.

For the reasons I set forth in my discussion of claim 1 above, an “upper layer application programming interface” that is enabled “by unified capabilities” to “program the plurality of network device applications and plurality of cloud applications independent of network device hardware and cloud device hardware” in Element [1.4] means an upper layer interface for holistic runtime³ programming of a distributed application as a whole, and where the unified interface abstracts away the heterogeneous characteristics of both the network device hardware and the cloud device hardware. The Petition fails to cite any portion of Vasell (Exs. 1004, 1015 and 1016) detailing an “upper layer application programming interface” that is enabled “by unified capabilities” to program distributed applications, as recited in Element [1.4].

99. The Petition alleges the an upper layer API can be found within nine (9) different passages of Vasell, which comprises Exs. 1004 and 1016 to the Petition. See, Petition, 55-57, citing Ex. 1016, 4, 6, 8-9; Ex. 1004, 2:60-3:1, 5:15-35, 22:35-37, 20:52-53, 16:59-60, FIG. 6. But none of the cited passages mention an upper layer API for programming distributed applications as claimed in Element [1.4]. Details for each of the passages are described below.

100. Petitioner's first citation in Vasell (Ex. 1004) states:

³ The use of "runtime" is meant to signify that the distributed applications are in a state where they are operating (or running).

"The service gateway system according to the present invention facilitates the development, implementation, operation and maintenance of services in an integrated manner so that the interface between different services providers and the end user is transparent to both. Each service may comprise a set of functionalities and logic which are implemented as software service applications. The service application may be distributed among various pieces of equipment which are geographically separated." Ex. 1004, 2:60-3:1.

This passage describes the high-level purpose and architectural benefits of the service gateway system, such as integrated service management and transparent interfaces between service providers and end-users. It discusses how service applications are implemented and distributed. However, it does not mention any specific programmatic interfaces, methods, functions, or protocols that would allow a user program distributed applications as claimed in the '823 Patent. This passage does not disclose an upper layer API to program distributed applications as claimed in the '823 Patent.

101. Petitioner's next citation in Vasell (Ex. 1004) states:

"Service gateways according to the present invention are preferably open platforms to allow for the development of service applications by third party developers. Moreover, standard communication protocols and programming languages may be used in order to facilitate such software service applications development. In addition to being open platforms, service gateways according to the present invention are intended to be horizontal in nature so that a plurality of service providers can share the same platform infrastructure." Ex. 1004, 3:10-19.

This passage emphasizes that service gateways are "open platforms" designed for "third party developers" and utilize "standard communication protocols and

programming languages." While it suggests an environment conducive to software development, it describes general design principles and enabling technologies (protocols, languages) rather than defining any specific interfaces for programming distributed applications. The mention of "standard communication protocols" refers to communication mechanisms, not a developer-facing upper layer API to program distributed applications as disclosed and claimed in the '823 Patent.

102. Petitioner's next citation in Vasell (Ex. 1004) states:

"The need for remote service gateway management arises due to the logistics of managing an extensive network of geographically dispersed service gateways. The service gateway network may potentially range from several hundred thousand to millions of service gateway units. At least some of the many service gateways will be installed in places not easily accessible. It would be logistically difficult to physically access all service gateways in order to, for instance, upgrade software, supervise operations, and detect errors. Therefore, the ability for a service gateway to be managed remotely in as many ways as possible is desirable. Another desirable characteristic of the service gateway design is that it be compatible to established communications standards, protocols, interface specifications and technologies. In addition to enhancing communication and interfacing among various systems, service gateway compatibility potentially opens third-party markets for service gateway hardware and services. However, specialized portions of the service gateway system may be specifically designed and developed." Ex. 1004, 5:15-35.

This extensive passage discusses the logistical challenges of remote service gateway management and the desirability of compatibility with "established communications standards, protocols, interface specifications and technologies."

It highlights the need for remote management capabilities and interoperability,

which could implicitly rely on interfaces. However, the passage describes the problems, desirable characteristics, and general requirements of the system's management and compatibility without defining any specific programmatic interfaces (APIs) for achieving these. The "interface specifications" mentioned refer to general communication standards, not any specific API (let alone an upper layer API) for programming distributed applications as disclosed and claimed in the '823 Patent.

103. Petitioner's next citation in Vasell (Ex. 1004) states:

"For example, as conceptually illustrated in FIG. 3, open service gateways according to the present invention may be used to integrate many independent service providers to further exploit efficiencies associated with using this same communication and information processing infrastructure." Ex. 1004, 10:18-23.

This passage describes the conceptual use of open service gateways for integrating service providers and leveraging infrastructure efficiencies. It discusses the architectural role and benefits of the system. However, it does not define or imply any specific programmatic interfaces for developers to utilize this integration or exploit efficiencies. It remains at a conceptual level of system function. Therefore, this passage does not disclose an API -- let alone an upper layer API for programming distributed applications as disclosed and claimed in the '823 Patent.

104. Petitioner's next citation in Vasell (Ex. 1004) states:

"The development of the boxlets 64-69 is not restricted to any particular software language or operating system. According to this

exemplary embodiment, the only code specific to development of the boxlets 64–69 is in class libraries 95 which contain the service application 70-72 program interfaces for the main services layer 100 and the system services layer 110." Ex 1004, 13:37-43.

This passage explicitly states that "class libraries 95... contain the service application 70-72 program interfaces for the main services layer 100 and the system services layer 110." These APIs are primarily low-level management APIs or infrastructure APIs. They define how "boxlets" (internal code components) interact with the foundational "main services layer" and "system services layer." The interfaces are for internal system communication and control, facilitating interactions with core services rather than providing upper layer functions for programming distributed applications as disclosed and claimed in the '823 Patent.

105. Petitioner's next citation in Vasell (Ex. 1004) states:

"If one of the cells 90-92 crashes too many times within a given predetermined time period, the cell manager 93 may disable it. The cell 90-92 remains disabled until it is either replaced as a result of loading a new cell table 94 or reenabled by the cell manager 93 or by the system management application program interface (API)." Ex. 1004, 16:55-60.

This passage explicitly mentions a "system management application program interface (API)" that can "re-enable" a disabled cell. This is a low-level management API. Its function is to control the lifecycle and state of internal system components ("cells"), specifically to re-enable them after a crash. This is a granular, administrative function focused on maintaining the operational integrity

of the system's internal elements. It is distinct from an upper layer API designed for programming network and cloud applications that form distributed applications as disclosed and claimed in the '823 Patent.

106. Petitioner's next citation in Vasell (Ex. 1004) states:

"These and other design aspects of the service gateway system ensure the security, isolation and robustness of the service applications 70-72, facilitate management of service applications 70-72 lifetimes, and allow resource management for cells 90-92." Ex. 1004, 20:49-54.

This passage describes general design objectives and benefits of the service gateway system, such as ensuring security, isolation, robustness, and facilitating lifetime and resource management for service applications and cells. The passage itself outlines goals and outcomes of the design rather than disclosing any specific programmatic interfaces, let alone upper layer APIs designed for programming network and cloud applications that form distributed applications, as disclosed and claimed in the '823 Patent.

107. Petitioner's next citation in Vasell (Ex. 1004) states:

"The management system service 76 provides an external interface through which the service applications 70-72 may be downloaded, installed, removed, executed, and controlled." Ex. 1004, 22:35-38.

This passage describes that "The management system service 76 provides an external interface through which the service applications 70-72 may be downloaded, installed, removed, executed, and controlled." This "external interface" functions as a lower layer management API. It exposes operations

related to the lifecycle and basic control of "service applications" (e.g., download, install, remove, execute). These are administrative tasks focused on the deployment and fundamental operation of software components within the system, rather than providing upper layer APIs for programming network and cloud applications that form distributed applications as disclosed and claimed in the '823 Patent.

108. Petitioner's next citation in Vasell (Ex. 1004) states:

"FIG. 6 depicts an exemplary organization of a software system for providing a connectivity based Service gateway according to the present invention." Ex. 1004, 4:16-18.

This passage merely provides a descriptive caption for FIG. 6, indicating that the figure depicts a software system organization. Notably absent from the depicted software system is an API. Accordingly, like every other portion of Vasell 1004 cited by the Petition, Fig. 6 fails to disclose an upper layer API for programming applications.

109. The Petition asserts that Vasell Ex. 1016 discloses an API for programming applications that aligns with such upper layer APIs. However, Vasell Ex. 1016 mentions APIs only twice, and neither instance describes an API functioning as an upper layer hardware-independent programming interface for distributed applications spanning network and cloud components. Instead, the disclosed APIs are focused on enabling local application interaction with internal

infrastructure or facilitating internal component development within the service gateway itself. The relevant passages from Vasell Ex. 1016 are set forth below, each followed by a rationale as for why it does not disclose upper layer APIs for programming applications as required by the '823 patent.

110. One of Petitioner's citations in Vasell (Ex. 1016) states:

"The development environment must follow the "write-once, run-everywhere" maxim and should be based on Java standards. New applications will interact with the e-service infrastructure through Java application program interfaces (API) that comply with mainstream Java development. By leveraging the Java development, the application software environment can be taken to a higher level of abstraction, allowing non specialists to develop service applications more easily." Ex. 1016, at 4.

This passage describes APIs that enable new applications to "interact" with the e-service infrastructure. The focus is on facilitating local deployment and execution of applications within the gateway's defined environment, allowing them to utilize the gateway's internal services and resources. While Java development offers abstraction for application portability, this API itself is not described as an upper layer programming interface that abstracts away underlying hardware specificities to program or control heterogeneous network device applications and cloud applications as a cohesive, distributed entity. In my opinion, using the syntax portability (allowing code to run on different hardware) of a high level language like Java as an API does not meet the "upper layer" requirements of the claim language. The comment in the above quotation relating to a "higher level" of

abstraction is simply a reference to what Java provides, and does not equate to an "upper layer" API as recited in Element [1.4]. As noted above, Vasell discloses device drivers which are closest to the hardware layer. The Petition does not explain why, simply because Java provides "a higher level of abstraction" than something else, does not mean that it provides the level of abstraction provided by an "upper layer API" as recited in Element [1.4].

111. Another one of Petitioner's citations in Vasell (Ex. 1016) states:

“Boxlets are created using a standard Java development environment. For instance, the Java development kit ODK from Sun Microsystems can be used as well as other development environments. The only parts that are specific to boxlet development are the libraries that contain APIs for the main services and system services layers.” Ex. 1016, at 8.

This passage explicitly states that "boxlets" (service application components) are created using a "standard Java development environment," indicating that the primary act of application programming relies on conventional development tools. The APIs mentioned are contained within class libraries which are at a low layer of Vasell's system, and are identified as "specific to boxlet development" for interacting with the gateway's "main services and system services layers." These are internal interfaces designed for modularity and interaction within a single service gateway device. They do not constitute upper layer APIs for programming applications in a hardware-independent manner across a distributed network and cloud infrastructure.

112. Vasell (Ex. 1016) describes the "e-box" as executing localized service logic and interacting with external service platforms or cloud infrastructure. However, there is no suggestion that these remote systems are programmed through the same upper layer APIs, nor that the remote and local software components form a distributed application programmed by such an API. Instead, cloud-side functionality is implemented as separate external services that communicate with the e-box using conventional network protocols. This indicates a system that communicates with external entities rather than programmatically controls them via an upper layer API.

113. The APIs disclosed in Vasell (Ex. 1016) are inherently localized to the gateway device. They enable internal application interaction, support the development of internal service components, and facilitate administrative tasks within the gateway. They do not extend to providing a hardware-independent, upper layer programming interface that encompasses both edge (gateway) and cloud components as part of a cohesive, distributed application. Accordingly, Vasell's APIs (Java APIs and internal main/system services APIs) do not "program" a plurality of network device applications and cloud applications as claimed, because they are (a) scoped to specific (cells, gate manager, system services), and (b) do not disclose a unified, upper-layer API layer used to program

both sides of a device/cloud distributed application in a unified, hardware-independent manner.

114. In summary, Vasell (Ex 1004 and Ex. 1016) fails to disclose or suggest APIs that function as "upper layer APIs" for programming applications across a distributed network and cloud environment, independent of hardware. The described interfaces are limited to facilitating localized application interaction or internal component development within the service gateway, which falls short of the hardware-agnostic programming of distributed applications required by the '823 Patent.

XI. Distinctions Between "Unified Capabilities" As Set Forth In Claim 1 Of the '823 Patent And Vasell

115. The element in claim 1 that recites "wherein the plurality of network device applications and plurality of cloud applications form *unified capabilities* enabling a plurality of upper layer application programming interfaces (APIs) to program the plurality of network device applications and plurality of cloud applications independent of network device hardware and cloud device hardware," establishes a stringent requirement for "unified capabilities" that go beyond mere connectivity. This mandate describes a sophisticated, abstract layer within a distributed software-defined networking (dSDN) architecture, aiming to create a consolidated programming environment for both network device applications and cloud applications through a single, consistent paradigm. This effectively merges

the control planes of the device and the cloud, normalizing heterogeneous network elements into a unified view that is independent of specific hardware. This design enables a high level of abstraction for network programmability, providing a cohesive environment for managing diverse applications and infrastructure elements.

116. This claim element highlights a foundational principle: a single, coherent set of "unified capabilities" that allows for programmatic control over both network and cloud applications without needing to account for the underlying hardware. This approach is designed to manage heterogeneous elements by normalizing them into a unified view.

117. Vasell (Ex 1004) frequently uses the term "integrated" to describe the internal coherence and physical bundling of its components. This integration focuses on combining hardware and software within a specific device to form a functional unit, rather than creating a unified, software-defined programming interface that spans a distributed network or merges control planes. Thus, the teachings of Vasell would not have taught a POSITA the unified capabilities in Claim 1 of the '823 patent.

118. The Petition alleges the unified capabilities can be found within nine (9) different passages of Vasell. See, Petition, 55-57, citing Ex. 1016, 4, 6, 8-9; Ex. 1004, 2:60-3:1, 5:15-35, 22:35-37, 20:52-53, 16:59-60, FIG. 6. But none of the

cited passages mention this type of capabilities as described below and as envisioned by the '823 Patent.

119. The first passage in Vasell (Ex. 1004) relied on for the claimed "unified capabilities" states:

"The service gateway system according to the present invention facilitates the development, implementation, operation and maintenance of services in an integrated manner so that the interface between different services providers and the end user is transparent to both. Each service may comprise a set of functionalities and logic which are implemented as software service applications. The service application may be distributed among various pieces of equipment which are geographically separated." Ex. 1004, 2:60-3:1.

120. The "integrated manner" described here refers to integrated service management and transparent service management within the gateway system. The integration is about making internal processes coherent and user-facing interactions seamless, not about providing unifying capabilities for distributed application programming. It does not teach a unified programming or control environment that abstracts away underlying, distinct hardware and software components into a single, hardware-agnostic paradigm for programmatic control across diverse distributed applications.

121. The next passage in Vasell (Ex. 1004) relied on for the claimed "unified capabilities" states:

"Service gateways according to the present invention are preferably open platforms to allow for the development of service applications by third party developers. Moreover, standard communication

protocols and programming languages may be used in order to facilitate such software service applications development. In addition to being open platforms, service gateways according to the present invention are intended to be horizontal in nature so that a plurality of service providers can share the same platform infrastructure." Ex. 1004, 3:10-19.

122. "Open platforms," "standard communication protocols," and a "horizontal" nature promote interoperability and third-party development by providing access points and common communication methods. Thus, they enable components to interact and developers to build applications for the platform, but they do not provide a single, consistent programming paradigm for simultaneously controlling heterogeneous network devices and cloud applications. The focus of Vasell remains on enabling communication and individual application deployment within defined system boundaries, rather than unifying capabilities of programming or control across heterogeneous network and cloud elements.

123. The next passage in Vasell (Ex. 1004) relied on for the claimed "unified capabilities" states:

"The need for remote service gateway management arises due to the logistics of managing an extensive network of geographically dispersed service gateways. The service gateway network may potentially range from several hundred thousand to millions of service gateway units. At least some of the many service gateways will be installed in places not easily accessible. It would be logistically difficult to physically access all service gateways in order to, for instance, upgrade software, supervise operations, and detect errors. Therefore, the ability for a service gateway to be managed remotely in as many ways as possible is desirable. Another desirable characteristic of the service gateway design is that it be compatible to

established communications standards, protocols, interface specifications and technologies. In addition to enhancing communication and interfacing among various systems, service gateway compatibility potentially opens third-party markets for service gateway hardware and services. However, specialized portions of the service gateway system may be specifically designed and developed." Ex. 1004, 5:15-35.

124. This passage addresses operational logistics such as remote management and interoperability through compatibility with "established communications standards, protocols, interface specifications and technologies." These capabilities are crucial for managing individual gateways and allowing them to communicate with external systems. However, they do not describe unified capabilities of programming or control across heterogeneous network and cloud elements.

125. The next passage in Vasell (Ex. 1004) relied on for the claimed "unified capabilities " states:

"For example, as conceptually illustrated in FIG. 3, open service gateways according to the present invention may be used to integrate many independent service providers to further exploit efficiencies associated with using this same communication and information processing infrastructure." Ex. 1004, 10:18-23.

126. The "integration" of service providers mentioned here is fundamentally infrastructural and operational. It refers to allowing multiple service providers to share a common underlying platform and its resources to achieve efficiency. It does not describe a unified programming interface or a single control

logic that abstracts away the underlying differences in how these independent service providers' applications are programmed or how their distributed components are controlled.

127. The next passage in Vasell (Ex. 1004) relied on for the claimed "unified capabilities " states:

"The development of the boxlets 64-69 is not restricted to any particular software language or operating System. According to this exemplary embodiment, the only code specific to development of the boxlets 64-69 is in class libraries 95 which contain the service application 70-72 program interfaces for the main services layer 100 and the system services layer 110." Ex 1004, 13:37-43.

128. The "class libraries 95" and "program interfaces" described here are specific to the internal development of "boxlets," which are internal code components implementing services within the gateway. These interfaces facilitate interaction between these internal components and the system layers. The interaction is limited to internal component interaction within a single device's framework and designed for internal plumbing and modularity, not for unifying control across disparate environments. It does not provide unified capabilities for a system-wide programming abstraction for distributed applications that spans heterogeneous network and cloud devices.

129. The next passage in Vasell (Ex. 1004) relied on for the claimed "unified capabilities " states:

"If one of the cells 90-92 crashes too many times within a given predetermined time period, the cell manager 93 may disable it. The cell 90-92 remains disabled until it is either replaced as a result of loading a new cell table 94 or reenabled by the cell manager 93 or by the system management application program interface (API)." Ex. 1004, 16:55-60.

130. The "system management application program interface (API)" discussed here serves a specific, localized administrative function: re-enabling a disabled internal "cell." This is an operational maintenance task focused on the stability and recovery of internal system components within a single service gateway. It does not constitute a unified programming environment that spans network and cloud components.

131. The next passage in Vasell (Ex. 1004) relied on for the claimed "unified capabilities " states:

"These and other design aspects of the service gateway system ensure the security, isolation and robustness of the service applications 70-72, facilitate management of service applications 70-72 lifetimes, and allow resource management for cells 90-92." Ex. 1004, 20:49-54.

132. This passage outlines fundamental system-level operational features such as security, isolation, robustness, and management of application lifetimes and resources. While essential for the stable and efficient functioning of the service gateway, these are localized operational controls critical to the individual gateway's reliability and internal integrity. They do not constitute unified capabilities for programming that abstracts away hardware dependencies and

provides a single, consistent interface for programming diverse, distributed elements (like network devices and cloud components) as a cohesive entity.

133. The next passage in Vasell (Ex. 1004) relied on for the claimed "unified capabilities " states:

"The management system service 76 provides an external interface through which the service applications 70-72 may be downloaded, installed, removed, executed, and controlled." Ex. 1004, 22:35-38.

134. This "external interface" for the "management system service" enables administrative lifecycle operations (downloading, installing, removing, executing, controlling) on "service applications" within the gateway. This is limited to managing applications on the local device. It does not teach "unified capabilities enabling a plurality of upper layer application programming interfaces (APIs)" of distributed applications.

135. The next passage in Vasell (Ex. 1004) relied on for the claimed "unified capabilities " states:

"FIG. 6 depicts an exemplary organization of a software system for providing a connectivity-based Service gateway according to the present invention." Ex. 1004, 4:16-18.

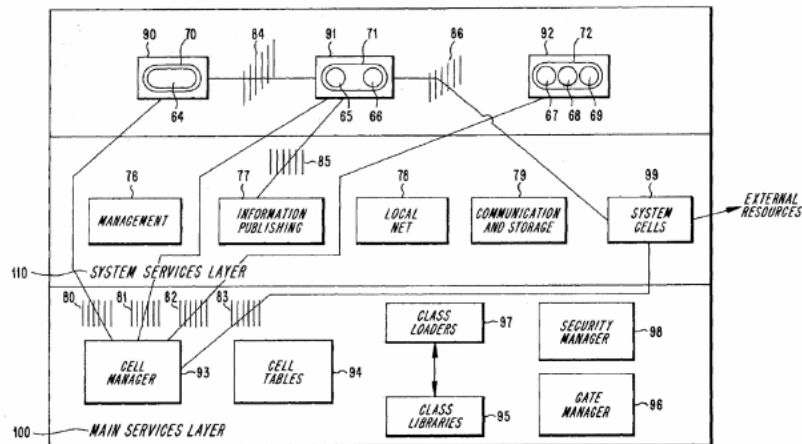


FIG. 6

136. This passage, referencing FIG. 6, describes the internal structural organization of the software system within a single service gateway. The figure illustrates how internal components are arranged and interact within the device for localized management and organization of services. Notably absent from the depicted software system is any disclosure of “providing unified capabilities enabling a plurality of upper layer application programming interfaces (APIs).”

137. Overall, Vasell describes a system where components are physically integrated to deliver services, and interfaces are used to manage software installation or facilitate specific links. It fundamentally lacks the "unified capabilities" that abstract these components into a single, consolidated programmable interface for control of a distributed application as a whole. There is no teaching in Vasell of a mechanism that normalizes disparate network and cloud APIs to offer a consistent programming model by providing unified

capabilities enabling a plurality of upper layer application programming interfaces (APIs), as required by the '823 Patent.

138. In my opinion, Petitioner's mapping effectively substitutes generic software portability for the specific "unified capabilities" architecture described by in the '823 Patent (HAL + framework libraries + access-controlled platform APIs + device/cloud coordination). (Ex. 1001, 11:54–60; Ex. 1001, 12:20–22; Ex. 1001, 12:25–51.)

139. As I mentioned in my discussion of claim construction, claim 1 of the '823 Patent recites an "upper layer application programming interface" that is enabled "by unified capabilities" to "program the plurality of network device applications and plurality of cloud applications independent of network device hardware and cloud device hardware," where the plurality of network device applications and plurality of cloud applications are defined earlier in the claim as forming "distributed applications. Collectively, these limitations signify an upper-layer interface designed for the holistic runtime programming of a distributed application as a whole. This unified interface effectively abstracts away the heterogeneous characteristics of both the network device hardware and the cloud device hardware, which may holistically be required by the applications running within the distributed system. Vasell's "external interface" (Ex. 1004) for the "management system service" enables administrative lifecycle operations

(downloading, installing, removing, executing, controlling) on "service applications" within the gateway is limited to managing applications on the local device -- rather than the holistic runtime programming of a distributed application as a whole. Nothing in Vasell's external interface (Ex. 1004) suggests an upper layer API for programming disparate network devices and cloud resources *as if they were a single programmable entity*, i.e., the holistic runtime programming of a distributed application *as a whole*.

140. The Petition asserts that Vasell Ex. 1016 discloses an API for programming applications that aligns with such unified capabilities. However, a review of Vasell Ex. 1016 reveals that it mentions APIs only twice, and neither instance describes a mechanism for providing unified, hardware-independent programming or control across diverse network devices and cloud components as a single entity. The descriptions focus on enabling application interaction with internal infrastructure or facilitating internal component development within the service gateway itself. Here are the relevant passages from Vasell Ex. 1016, each followed by a rationale as to why it does not disclose or suggest unified capabilities as required by the '823 patent.

141. The Petition relies on the following passage in Vasell (Ex. 1016) for the claimed "unified capabilities " states:

"The development environment must follow the "write-once, run-everywhere" maxim and should be based on Java standards. New

applications will interact with the e-service infrastructure through Java application program interfaces (API) that comply with mainstream Java development. By leveraging the Java development, the application software environment can be taken to a higher level of abstraction, allowing non- specialists to develop service applications more easily.” Ex. 1016, at 4.

142. This passage describes APIs that enable new applications to interact with the e-service infrastructure, promoting ease of development through Java standards and abstraction. However, this interaction is about facilitating the local deployment and execution of applications within the gateway's defined e-service environment. It does not teach “unified capabilities enabling a plurality of upper layer application programming interfaces (APIs) ” to program distributed applications, as claimed in the ‘823 Patent. As mentioned above, the limitations in claim 1 signify an upper-layer interface designed for the holistic runtime programming of a distributed application as a whole. It effectively abstracts away the heterogeneous characteristics of both the network device hardware and the cloud device hardware, which may holistically be required by the applications running within the distributed system. The Java APIs disclosed in Vasell that enable new applications to interact with the e-service infrastructure do provide *holistic* runtime programming of a distributed application *as a whole*.

143. Technically, “write once/run everywhere” is language portability (e.g., JVM portability). It does not, by itself, provide a unified abstraction layer that (i) normalizes heterogeneous infrastructure, and (ii) exposes platform-level

APIs for infrastructure programmability independent of underlying vendor hardware.

144. The Petition relies on the following passage in Vasell (Ex. 1016) for the claimed "unified capabilities " states:

“Boxlets are created using a standard Java development environment. For instance, the Java development kit JDK from Sun Microsystems can be used as well as other development environments. The only parts that are specific to boxier development are the libraries that contain APIs for the main services and system services layers.” Ex. 1016, at 8.

145. This passage refers to APIs contained within class libraries, which are specific to the development of "boxlets"—internal service application components designed to interact with the gateway's "main services and system services layers." These APIs facilitate internal component interaction and modularity within a single service gateway device. The scope is confined to internal plumbing and development within the gateway's localized framework, not to unifying control across disparate environments to provide “Unified capabilities enabling a plurality of upper layer application programming interfaces (APIs) to program the plurality of network device applications and plurality of cloud applications," as claimed in the ‘823 patent.

146. Vasell Ex. 1016 describes a service gateway, the "e-box," that executes localized service logic in the form of Java-based "boxlets." While the e-box may interact with external service platforms or cloud infrastructure, there is

no suggestion that those remote systems are programmed through a unified, hardware-independent paradigm. Instead, the cloud-side functionality appears to be implemented as external services that communicate with the e-box using conventional network protocols. This approach facilitates communication between separate entities rather than merging their control planes into a single, programmable unit that abstracts away hardware specifics across the entire distributed system. Thus, it does not teach “unified capabilities enabling a plurality of upper layer application programming interfaces (APIs) to program the plurality of network device applications and plurality of cloud applications,” as recited in the '823 Patent.

147. Vasell Ex. 1016's architecture remains tightly coupled to the characteristics of the underlying physical hardware. The programming environment does not abstract away device-specific interfaces. Vasell Ex. 1016 does not describe unified capabilities that allows for programming applications independent of network device hardware and cloud device hardware. Instead, the system's compatibility with diverse LAN technologies implies the necessity of adapting to or integrating with device-specific interfaces for each protocol.

148. The APIs described in Vasell Ex. 1016 are confined to the gateway device and do not teach “Unified capabilities enabling a plurality of upper layer application programming interfaces (APIs) to program the plurality of network

device applications and plurality of cloud applications,” as recited in the '823 Patent. They are designed for localized governance and operation of the gateway's internal services and applications, enabling administrative control and internal component interaction.

149. In summary, Vasell Exs. 1004 and 1016 do not disclose or suggest APIs that enable unified capabilities in the sense of providing a single, consistent programming paradigm for holistically controlling the network device applications and cloud applications that form distributed applications independently of underlying hardware. Accordingly the petition has not demonstrated that Vasell discloses “Unified capabilities enabling a plurality of upper layer application programming interfaces (APIs) to program the plurality of network device applications and plurality of cloud applications” required by Claim 1 of the '823 Patent.

150. For the foregoing reasons, it is my opinion that Claim 1 of the '823 patent would not have been obvious to POSITA over the prior art relied on in the Petition.

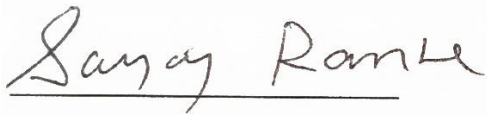
151. Since Claims 2-5, 7-8, and 12-15 are dependent claims corresponding to Claim 1, they would not have been obvious over the prior art in Petitioner's Petition for *inter partes* review for the same reasons as Claim 1.

152. Ground 2 challenges dependent claims 3-5, 7-8 and 18 over Vasell, Alves, Rellermeyer and Hall. However, all of the mappings in Ground 2 are predicated on the mapping for claim 1 in Ground 1 which, as discussed above, fails to satisfy the claim language. Thus, Ground 2 also fails to demonstrate the unpatentability of dependent claims 3-5, 7-8 and 18.

XII. Certification

153. All statements made herein of my own knowledge are true, and all statements made on information and belief are believed to be true. Further, I am aware that these statements are made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. § 1001. I declare under penalty of perjury that the foregoing is true and correct.

Executed on December 23, 2025, in Gainesville, Florida.

A handwritten signature in cursive script that reads "Sanjay Ranka". The signature is written in black ink on a light-colored background. A horizontal line is drawn underneath the signature.

Dr. Sanjay Ranka