

UNITED STATES PATENT AND TRADEMARK OFFICE

---

BEFORE THE PATENT TRIAL AND APPEAL BOARD

---

MICROSOFT CORPORATION

Petitioner,

v.

EDGE NETWORKING SYSTEMS, LLC,

Patent Owner.

---

Case No. IPR2025-00618  
U.S. Patent No. 11,695,823

---

**PATENT OWNER'S RESPONSE**

## TABLE OF CONTENTS

PATENT OWNER’S EXHIBIT LIST .....	vi
I. INTRODUCTION.....	1
II. THE SPECIFICATION OF THE ‘823 PATENT .....	5
A. Background .....	5
B. Overview Of Distributed Application Environment Disclosed In ‘823 Patent .....	5
C. Virtual Fabric For Communicating Between The FxCloud App Component and FxDevice Component Of A Distributed Application.....	8
D. Software Layers Of Platform Architecture .....	9
E. APIs For Communications Between Different Distributed Applications Or With Entities Outside Of The dSDN Platform .....	12
F. Examples Of Upper Layer APIs Disclosed In The ‘823 Patent .....	13
III. THE PROSECUTION HISTORY OF THE ‘823 PATENT .....	15
IV. THE CLAIMS OF THE ‘823 PATENT .....	17
A. The Plain And Ordinary Meaning Of “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” .....	18
B. The Petition Has Failed To Explain How It Is Interpreting The Claims .....	22
C. Response To The Directions From The Board .....	24
V. STATE OF THE ART - JAVA DEVELOPMENT ENVIRONMENT .....	25
VI. MATERIALS RELIED ON IN THE PETITION .....	29
A. U.S. Provisional Application 60/123,971 by Vasell et al. (Ex. 1016) .....	29
B. U.S. Patent No. 6,496,575 to Vasell (Ex. 1004) .....	32

C.	U.S. Provisional Patent Application No. 60/088,437 (Ex. 1015)	34
D.	“OSGi in Depth” by Alves (Ex. 1008)	34
F.	“Dependability as a cloud service - a modular approach” by Rellermeyer (Ex. 1011)	35
VII.	A PERSON HAVING ORDINARY SKILL IN THE ART	35
VIII.	THE PETITION FAILS TO MEET ITS BURDEN UNDER GROUND 1 TO ESTABLISH THAT THE CHALLENGED CLAIMS WOULD HAVE BEEN OBVIOUS	35
A.	Ground 1 Fails To Demonstrate That The Relied On References Meet All Of The Limitations Of Claim 1	35
1.	The Limitations Of Claim 1	36
2.	The Petition Fails To Demonstrate That Element [1.4] Is Met By Vasell	37
a)	Vasell’s “Write-Once, Run-Everywhere” Maxim And Where Boxlets Interact With The E-Service Infrastructure Through Java Application Program Interfaces Fails To Meet Element 1.4	37
b)	Vasell’s Disclosures Relating To System Software Fail To Meet Element 1.4	41
c)	Vasell’s Disclosures Relating To Service Applications And Boxlet Development Fail To Meet Element 1.4	42
d)	Vasell’s Disclosures Relating To Gates Operating In A Java Virtual Machine Fail To Meet Element 1.4	44
e)	Vasell’s Disclosures Relating To Its Service Gateway System Fail To Meet Element 1.4	45
f)	Vasell’s Disclosures Relating To Program Interfaces For The Main Services Layer 100 and The System Services Layer 110 System Fail To Meet Element 1.4	45

g).....Vasell’s Disclosures Relating To Its System Management Program Interface Fail To Meet Element 1.4.....	46
h)..... Vasell’s Disclosures Relating To Its Management System Service 76 Fails To Meet Element 1.4.....	48
3. The “Five Reasons” In The Petition Fail To Demonstrate That Element [1.4] Is Met By Vasell.....	49
a) Reason #1 - Vasell’s Alleged Disclosure Of Java Service Applications Are “Naturally Modular” ...	50
b) Reason #2 - Vasell’s Disclosure Of Standardized Communication Protocols .....	51
c)..... Reason #3 - Vasell’s Disclosure Of Standard Java APIs.....	51
d)..... Reason #4 - Vasell’s Alleged Disclosure Of Flexible, Transparent, And Standards-Compatible Interfaces For Lifecycle management .....	54
e)..... Reason #5 - Vasell’s Alleged Disclosure Of Standard Development And Management APIs Are Usable Regardless Of The Respective Service Platform Servers’ Hardware And Remote (Cloud) Servers’ Hardware On Which The Respective JVMs Run On The Respective OSs .....	56
4. Vasell’s “External Interface” Does Not Expressly Or Inherently Disclose A Plurality Of Application Programming Interfaces For Programming The Plurality Of Network Device And Cloud Applications .....	57
B. Petitioner Fails to Establish a Motivation to Combine or a Reasonable Expectation of Success .....	62
1. Petitioner’s proposed combination requires wholesale redesign, not routine “implementation details” .....	62
2. The combination is driven by hindsight because it presupposes the claimed “unified capabilities” objective ....	63

3. Petitioner also fails to prove a reasonable expectation of success for its re-architecture.....	64
C. Ground 1 Fails To Demonstrate That The Relied On References Meet All Of The Limitations Of Claims 2, 12-15 and 19 .....	65
IX. THE PETITION FAILS TO MEET ITS BURDEN UNDER GROUND 2.....	66
X. CONCLUSION .....	66

## PATENT OWNER'S EXHIBIT LIST

Ex. 2001	U.S. Patent Publ. 2013/0054763 (Van der Merwe)
Ex. 2002	<i>Ruckus Wireless, Inc. v. Hera Wireless SA et al.</i> , IPR2018-01739, Petition for Inter Partes Review, Paper 1,
Ex. 2003	Expert Declaration of Dr. Sanjay Ranka
Ex. 2004	CV of Dr. Sanjay Ranka
Ex. 2005	D. Friedel, Jr. and A. Potts, <i>JAVA Programming Language Handbook</i> , 1996
Ex. 2006	D. Kramer, <i>The Java Platform, A White Paper</i> , Sun Microsystems, 1996
Ex. 2007	U.S. Patent Publ. No. 2004/0107417
Ex. 2008	U.S. Patent No. 8,495,611
Ex. 2009	U.S. Patent Publ. No. 2008/0209491
Ex. 2010	U.S. Patent No. 7,630,706
Ex. 2011	U.S. Patent Publ. No. 2011/0265077
Ex. 2012	U.S. Patent Publ. No. 2012/0066665
Ex. 2013	U.S. Patent Publ. No. 2012/0266156
Ex. 2014	U.S. Patent Publ. No. 2013/0007254
Ex. 2015	U.S. Patent Publ. No. 2013/0263104
Ex. 2016	U.S. Patent Publ. No. 2014/0052976

## I. INTRODUCTION

The Petition challenges claims 1-5, 7-8, 12-15 and 18-19 (the “Challenged Claims”) of U.S. Patent No. 11,695,823 (the “ ‘823 Patent”) on two Grounds. As explained below, the Petition fails to demonstrate by a preponderance of the evidence that the relied on references meet all of the limitations of the Challenged Claims.

Claim 1 of the ‘823 Patent is representative and recites:

- [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.

Claim 1 includes several interrelated limitations. Starting with Element [1.3], the claim language provides that “the plurality of network applications” and “the plurality of second network applications” form “distributed applications.” Thus, “the plurality of first network applications” and “the plurality of second network applications” referenced later in the claim in Element [1.4] are not disparate,

unconnected applications but rather are parts of “distributed applications.” Ex. 1001, 10:25-31.

Element [1.4] further recites “a plurality of upper layer application programming interfaces (APIs)” used to program “the plurality of network device applications and plurality of cloud applications” which, as specified in Element [1.3], are components of the distributed applications. In 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 from the ‘823 Patent include APIs positioned at the highest software layer in the stack, just under the applications themselves. *See, id.*, Fx-Framework Libs 428 in Figure 4.

Element [1.4] goes on to recite that the “plurality of upper layer application programming interfaces (APIs)” function to “program the plurality of network device applications and plurality of cloud applications” (that form the distributed applications) in a manner that is “independent of network device hardware and cloud device hardware.” In the disclosed embodiments of the ‘823 Patent, the claimed independence from the network device hardware and cloud device hardware is provided at the upper layer APIs by, *inter alia*, abstraction layers below the upper layer APIs. *See, id.*, tools 416, native daemons 418, native libraries 420 and Hardware Abstraction Layers (HAL) 422 in Fig. 4, 12:18-21 and 13:36-37.

Element [1.4] also states 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” that form the distributed applications. As evidenced by the examples in the specification, the “unified capabilities” enabled by the upper layer APIs provide the user with the capability to program the distributed application as a 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 comprises the cloud application component. See, Section II(F), *infra*.

Referring back to Element [1.3], that element also 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. This claim language necessarily requires that the plurality of network device applications and the plurality of cloud 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. By referring back to “the plurality of network device and plurality of cloud applications,” that are in fact

operating (or running), Element [1.4] makes clear that the unified capabilities enable a plurality of upper layer APIs “to program” applications that are in an operating state. In other words, the claim language requires the use of upper layer API’s to program distributed applications while such applications are operating (or running). This aspect of the claim language corresponds to examples from the patent specification discussed in Section II(F), *infra*.

When the interrelated requirements described above are put together, 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” as recited in Element [1.4] means an upper layer interface for holistic runtime<sup>1</sup> programming of a distributed application as a whole, where the unified interface abstracts away the heterogeneous characteristics of both the network device hardware and the cloud device hardware. Section IV, *infra*.

Both Grounds in the Petition rely solely on Vasell for Element [1.4]. As explained in detail below, nothing in Vasell expressly or inherently discloses an “upper layer application programming interface” that is enabled “by unified

---

<sup>1</sup> The use of “runtime” is meant to signify that the distributed applications are in a state where they are operating (or running).

capabilities” to “program the plurality of network device applications and plurality of cloud applications independent of network device hardware and cloud device hardware,” as recited in Element [1.4].

For at least these reasons, the Petition fails to demonstrate that the Challenged Claims are unpatentable.

## **II. THE SPECIFICATION OF THE ‘823 PATENT**

### **A. Background**

The ‘823 Patent relates to cloud computing and dates back to 2013. At that time, “[e]xisting network solutions [were] built on proprietary hardware and software” that resulted in network operators being “unable to add new and customized features or capabilities.” Ex. 1001, 1:20-26. In these systems, network operators who wanted to “add new desired features” were required to make “requests to their infrastructure vendors or pursu[e] standardization processes.” *Id.*, 1:25-29. These approaches were “time consuming and resource intensive.” *Id.*, 1:29-31.

### **B. Overview Of Distributed Application Environment Disclosed In ‘823 Patent**

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).” *Id.*, 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.

“A high-level overview of a dSDN system 300 is depicted in FIG. 3,” which is reproduced with colored annotations added by Patent Owner. *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.

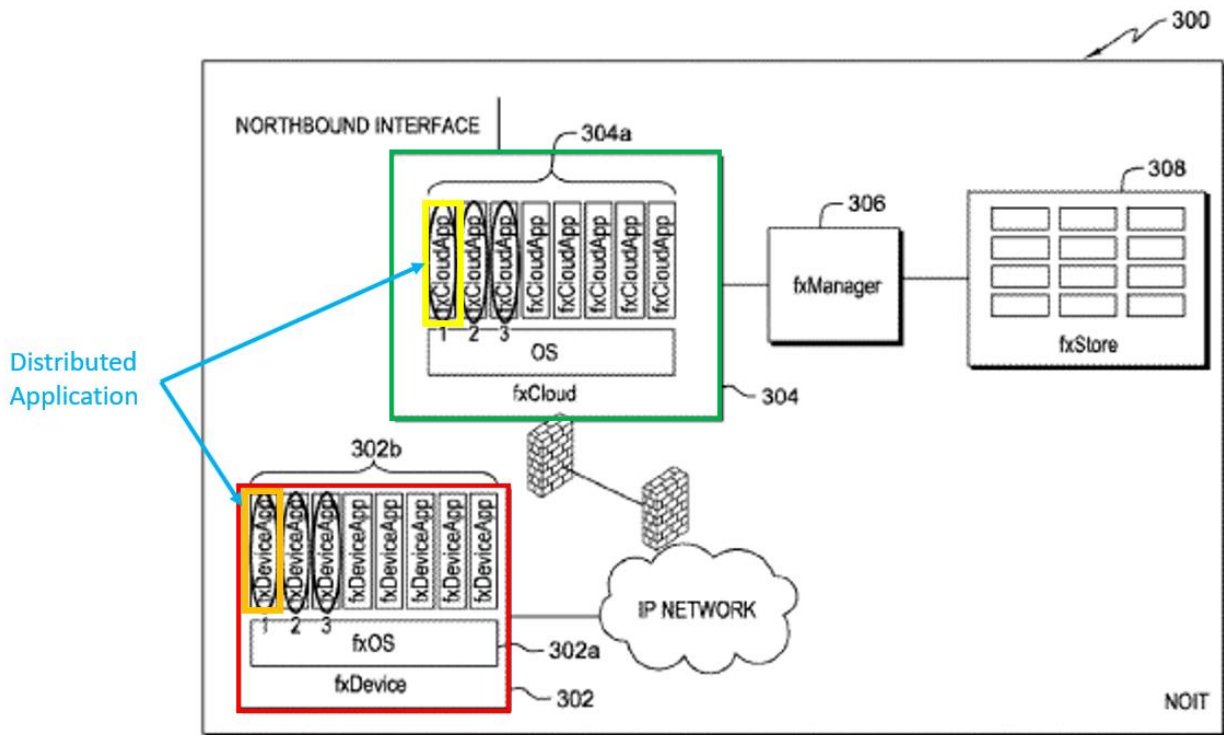


FIG. 3

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; Ex. 2003, ¶28.

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).

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. Ex. 2003, ¶30.

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

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.

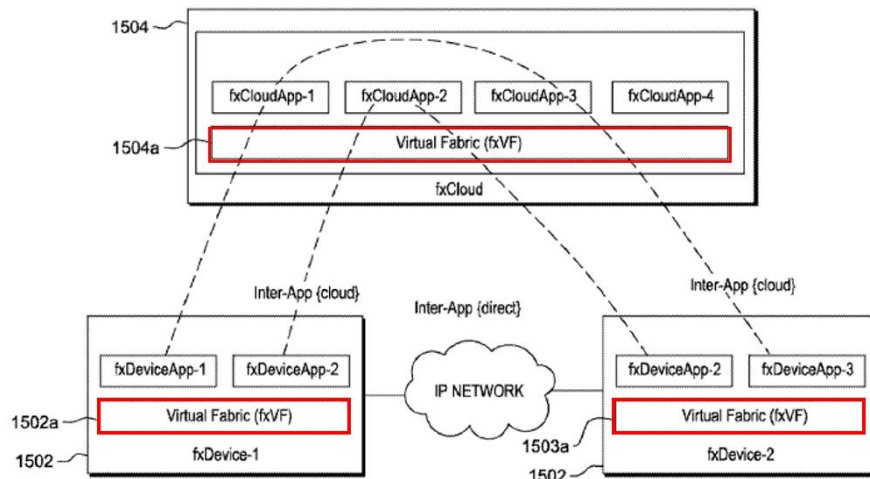


FIG. 15

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”). Ex. 2003, ¶¶31-32.

#### **D. Software Layers Of Platform Architecture**

“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. Ex. 2003, ¶33. 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.

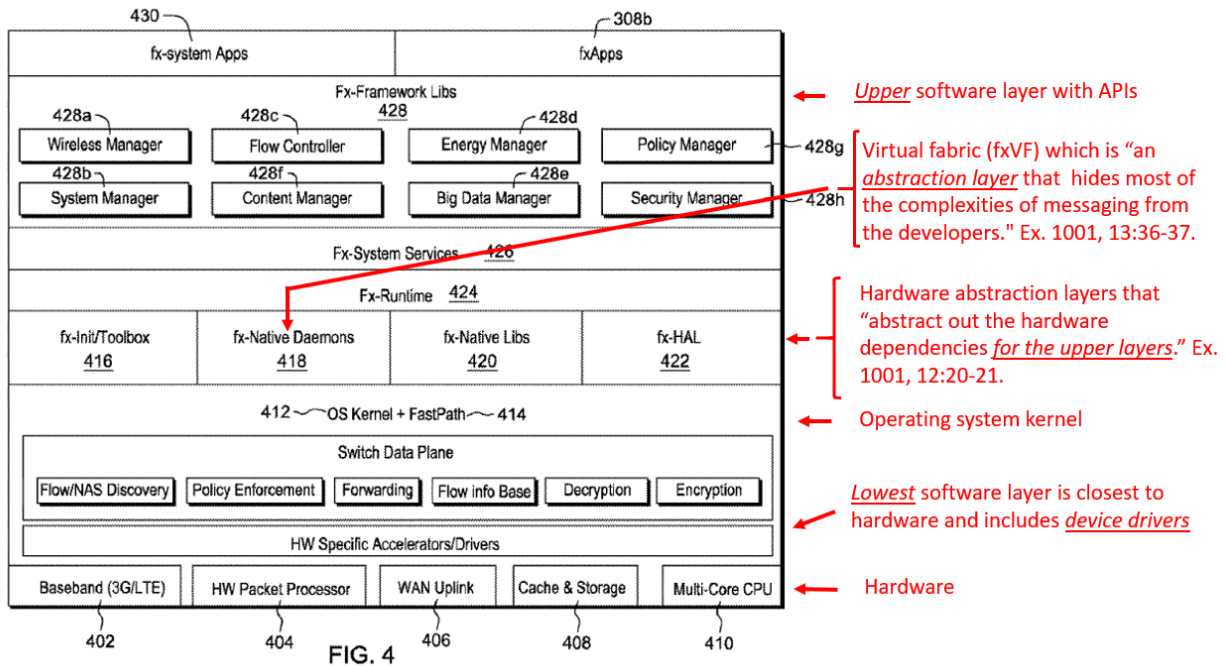


FIG. 4

“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.*” Ex. 2003, ¶34. 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 504” 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; Ex. 2003, ¶34.

Finally, the fx-Framework Library 428 is positioned at an *upper layer* of the software stack and consists of several Frameworks which are examples of the “*upper layer*” APIs recited in claim 1. Ex. 2003, ¶35. 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 428*a*: manipulation, monitoring, and configuration of the wireless interfaces; ...

Big Data Manager 428*e*: 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 428*f*: 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 428*g*: responsible for execution and implementation of polices 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.

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*”). Ex. 2003, ¶36.

#### **E. APIs For Communications Between Different Distributed Applications Or With Entities Outside Of The dSDN Platform**

The ‘823 Patent discloses 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). *Id.*, 21:23-27 and 28:49-51. In addition, the ‘823 Patent discloses a framework in which custom APIs are presented (or exposed) for communication between a distributed application and entities “outside the dSDN system.” Ex. 2003, ¶37. 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 dSND system*. *Id.*, 28:49-54 (emphasis added).

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):

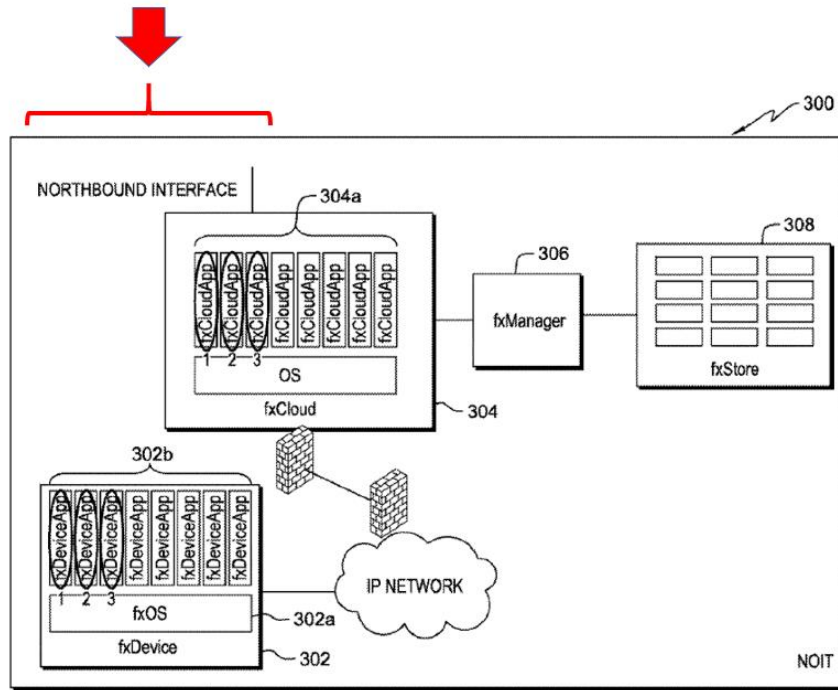


FIG. 3

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. Ex. 2003, 38.

**F. Examples Of Upper Layer APIs Disclosed In The ‘823 Patent**

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 (supporting an operating cellular network), so as to limit access to the cellular network in a case of national security. Ex. 2003, ¶39. 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.

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.*; Ex. 2003, ¶40. A POSITA would have understood that these custom APIs were intended to be accessed by the cellular service provider when the distributed application implementing the cellular service was already operating or running, in order to program the cellular service application to provide law enforcement with priority to the cellular network. Ex. 2003, ¶41.

### **III. THE PROSECUTION HISTORY OF THE '823 PATENT**

Following a Final Rejection of original claims 1-18 based on a nonstatutory double patenting rejection, the Applicant cancelled all of the previous claims and added new claims 19-38 (of which claims 19, 37 and 38 were independent). Ex. 1002, 124-129 and 133-137. Independent claim 19 recited in part:

19. (New) A system comprising a plurality of network elements, said network elements comprising: ...

wherein the programmable network device and programmable cloud device form unified capabilities enabling a plurality of upper layer application programming interfaces (APIs) that allow for simultaneous programming of the programmable network device and programmable cloud device independent of network device hardware and cloud device hardware. *Id.*, 125 (emphasis added).

In contrast to new claim 19, new independent claims 37 and 38 lacked the limitations in claim 19 specifying “unified capabilities enabling upper layer application programming interfaces (APIs) to program” the distributed applications. Ex. 1002, 128-129.

In the next Office Action, the Examiner rejected independent claims 37 and 38 as anticipated by U.S. Patent Publ. 2013/0054763 (Van der Merwe) but found independent claim 19 allowable over the same reference. Ex. 1002, 94-100. The Examiner found that Van der Merwe disclosed multiple applications on different hardware that “are in secure communication to form a distributed application” -- a requirement common to each of independent claims 19, 37 and 38.<sup>2</sup> Ex. 1002, 98-99. However, the Examiner found that Van der Merwe failed to suggest the claimed “unified capabilities enabling a plurality of upper layer application programming interfaces (APIs) to program” network device and cloud device applications that “are” (present tense) in secure communication with each other. *Id.*, 100 (reasons for allowance of claims 19-36).

The Van der Merwe reference before the Examiner during prosecution disclosed the use of APIs to develop applications. For example, Van der Merwe

---

<sup>2</sup> Patent Owner does not concede that the Examiner’s findings as set forth herein are correct.

disclosed the following:

To enable client administrators to provide coupling information and/or *create*, modify, and/or deploy *enterprise applications*, the example client coupler 130 includes an API interface 708. The example API interface 708 includes a web-based interface that enables client administrators to specify which VPMNs are to be coupled to which client VPNs and/or virtual machines. ...

The example API interface 708 of FIG. 7 also enables client administrators to develop, deploy, and/or modify enterprise applications through a MPaaS. Ex. 2001, ¶¶[0119]-[0120] (emphasis added).

Thus, while Van der Merwe disclosed the general use of APIs in developing applications, the Examiner understood that the general use of APIs to develop an application by itself was insufficient to meet the requirements of the allowed claims.

#### **IV. THE CLAIMS OF THE '823 PATENT**

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.

**A. The Plain And Ordinary Meaning Of “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”**

Several aspects of the plain and ordinary meaning of claim 1 are worthy of note.

First, Element [1.2] makes clear that “*the* plurality of network applications” and “*the* plurality of cloud applications” form “*distributed applications.*” Thus, as referenced throughout the claim, “*the* plurality of network applications” and “*the* plurality of cloud applications” are not disparate, unconnected applications but rather are parts of “*distributed applications.*” 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; Ex. 2003, ¶59a.

Second, Element [1.4] recites “a plurality of *upper layer* application programming interfaces (APIs).” As discussed above, 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); Ex. 2003, ¶59b.

Third, Element [1.4] 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, 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; Ex. 2003, ¶59c.

Fourth, Element [1.4] 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. 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 comprises 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(F), *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 comprises the cloud application component. Ex. 2003, ¶59d.

Fifth, Element [1.3] 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. This claim language necessarily requires that the plurality of network device applications and the plurality of cloud 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. By referring back to “the plurality of network device and plurality of cloud applications,” 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).<sup>3</sup> Ex. 2003, ¶59e.

Finally, Element [1.4] recites upper layer APIs that are enabled by unified capabilities “to program” the plurality of network device applications and plurality

---

<sup>3</sup> Element [1.4] states that “the plurality of network device applications and plurality of cloud applications ... form unified capabilities.” Here, “the plurality of network device applications and plurality of cloud applications” are the subject of the active verb “form.” The claim does not say that “the plurality of network device applications and the plurality of cloud device applications” that “are in secure communication with each other” are formed by unified capabilities (where “form” is used in the passive voice) -- but rather requires that “the plurality of network device applications and the plurality of cloud device applications” “in secure communication with each other” are themselves what “form unified capabilities.” Clearly, “the plurality of network device applications and plurality of cloud applications” recited in the beginning of Element [1.4] must have already been developed and exist before such applications could “form” unified capabilities, as claimed. This aspect of the final wherein clause is therefore consistent with the earlier recitation that the plurality of network device applications and the plurality of cloud device applications “are in secure communication with each other.”

of cloud applications forming the distributed applications which, as defined earlier in the claim, are in a state where they are operating (or running). Thus, the claim language requires the use of upper layer API's to program distributed applications while such applications are operating (or running). This aspect of the claim language corresponds, for example, to the example disclosed in the '823 Patent which, upon the occurrence of a national security event, enables an administrator of an operating cellular network to programmatically change 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. Ex. 2003, ¶59e.

**B. The Petition Has Failed To Explain How It Is Interpreting The Claims**

The Petition alleges that “no specific construction of any claim term is required because the prior art relied on in this Petition meets each of the claim terms under any reasonable construction.” Petition, 10 (emphasis added). But with respect to Element [1.4], the Petition never articulates what it contends “upper layer,” “unified capabilities,” or “to program” mean in the context of the '823 Patent, nor does it identify the claim meaning that purportedly renders its mappings correct. Instead, Petitioner proceeds as if it can avoid that analysis by invoking the conclusory refrain that its prior art meets the claims “under any reasonable construction.” That is not a substitute for Petitioner’s burden to prove unpatentability

by a preponderance of the evidence. Whatever discretion the Board may exercise in deciding whether construction is necessary to resolve a dispute, Petitioner must still present a coherent theory of unpatentability that applies the prior art to the claim language as understood in view of the intrinsic record. Here, Petitioner's obviousness theories for Element [1.4] depend on an unstated, outcome-driven reading of "upper layer," "unified capabilities," and "to program" that collapses these limitations into generic development interfaces and generic software management. That approach is inconsistent with the intrinsic record, and insufficient to satisfy Petitioner's burden for purposes of a final written decision.

In any event, Petitioner's decision to rest on the assertion that the art meets the claims "under any reasonable construction" has a concrete consequence: Petitioner must prevail under all constructions that are reasonably supported by the claim language and the intrinsic record, including the reasonable interpretations Patent Owner advances. Put differently, if there are multiple reasonable interpretations of "upper layer," "unified capabilities," or "to program" in Element [1.4], Petitioner fails to carry its burden if the asserted prior art does not satisfy even one of those reasonable interpretations.

As explained above, Patent Owner proffers the proper interpretations of "upper layer," "unified capabilities," and "to program" in Element [1.4] based on the claim language and the '823 Patent's disclosure. At a minimum, those

interpretations are reasonable. And, as shown below, the asserted prior art fails to meet Element [1.4] under these interpretations of “upper layer,” “unified capabilities,” and “to program.”

### **C. Response To The Directions From The Board**

In the Institution Decision, the Board stated:

... To the extent Patent Owner’s arguments for patentability depend on the scope of the term “program,” Patent Owner should assert its claim construction arguments and evidence in its Patent Owner Response or otherwise during trial, as permitted by our rules.

... To the extent Patent Owner’s arguments for patentability depend on the scope of the term “unified capabilities,” Patent Owner should assert its claim construction arguments and evidence in its Patent Owner Response or otherwise during trial, as permitted by our rules.

Institution Decision, 20-21.

In response, Patent Owner respectfully refers the Board to the discussion in Section IV(A), *supra*, with respect to the limitation in claim 1 that recites “*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.” For the reasons set forth in Section IV(A), in the context of the specification of the ‘823 Patent and claim 1 as a whole, it is Patent Owner’s position that, 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 runtime*<sup>4</sup> 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. It is also Patent Owner's position that, in Element [1.4], 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). Ex. 2003, ¶59e-f.

## V. STATE OF THE ART - JAVA DEVELOPMENT ENVIRONMENT

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.<sup>5</sup> The standard Java Platform included “classes” and related “APIs.” A

---

<sup>4</sup> The use of “runtime” is meant to signify that the distributed applications are in a state where they are operating (or running).

<sup>5</sup> 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],

Java class defines the data (variables, also known as fields or attributes) and methods (functions that operate on that data) for objects in a class. Ex. 2003, ¶65. A class did not represent a physical entity itself but rather provided the specifications for 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. Ex. 2003, ¶65.

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

---

[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).

language, utility, I/O, network, GUI, and applet services,” and was therefore implemented at a *low layer* close to the hardware. Ex. 2006, 14; Ex. 2003, ¶66.

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”). Ex. 2003, ¶67. A graphical representation of these operations is depicted below:

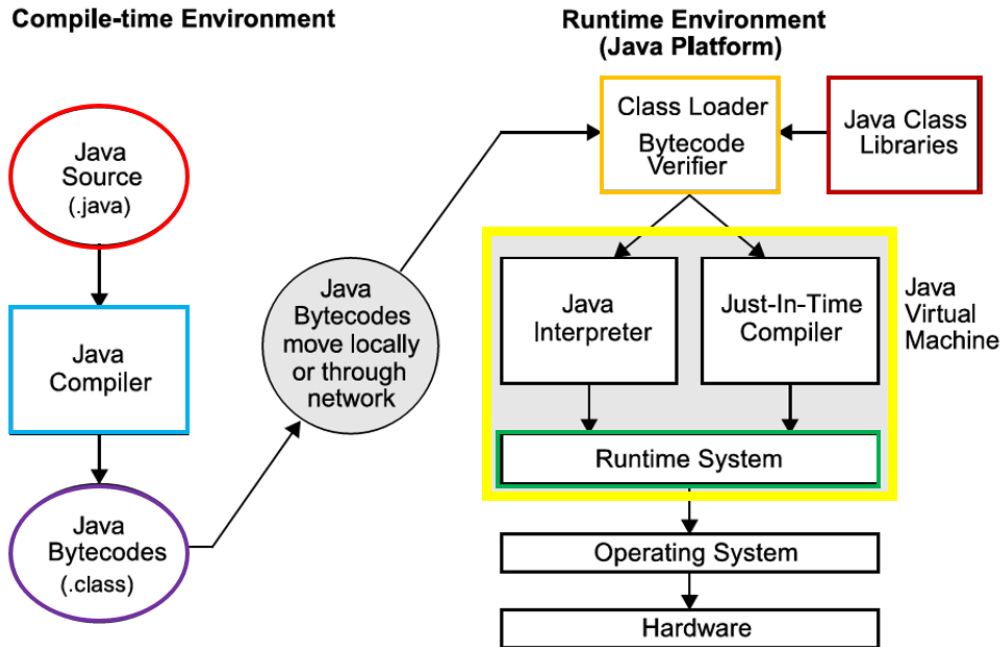


Figure 3. Source code is compiled to bytecodes, which are executed at runtime.

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. Ex. 2003, ¶68.

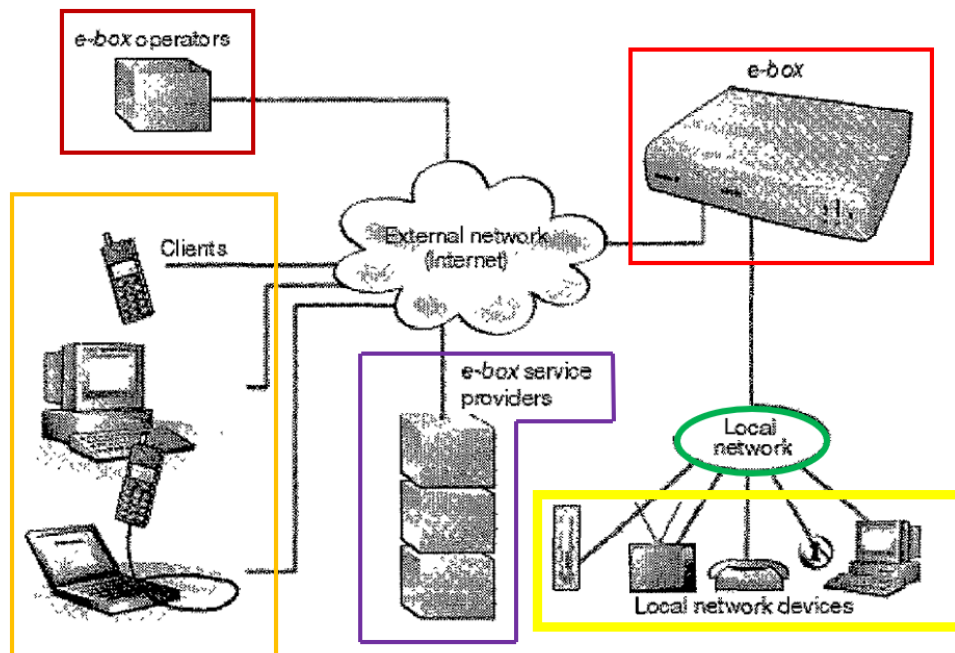
There is a critical distinction between the Java APIs used to write source code (as described above) and the APIs recited in the final wherein clause of the ‘823 Patent. Specifically, the Java APIs were used to write the original source code (red box) underlying the application itself, rather than to program the application *after* it was operational in the Runtime System (green box). Ex. 2003, ¶69.

## VI. MATERIALS RELIED ON IN THE PETITION

### A. U.S. Provisional Application 60/123,971 by Vasell et al. (Ex. 1016)

U.S. Provisional Application No. 60/123,971 (the “Vasell Provisional Application”) was entitled “Ericsson’s e-box system - An electronic services enabler.” Ex. 1016, 3. The Vasell Provisional Application presented “the design and architecture of the *e-box* system, the central part of which is a remote-controlled edge server to be installed in the home.” *Id.*, 1. “Ericsson’s new *e-box* system” was “developed to provide an efficient means of delivering complete e-services all the way to the home. *Id.* The authors envisioned a system that was “based on Java standards” in which “[n]ew applications will interact with the e-service infrastructure through Java application program interfaces (API) that comply with mainstream Java development.” *Id.* (emphasis added).

Figure 2 of the Vasell Provisional Application (reproduced below with colored annotations added) disclosed the architecture of the e-box system. *Id.*, 5. The system included “an edge server (the *e-box*) for installation in the home” (red box). *Id.*, 6. “The *e-box* is positioned between an external Internet protocol (IP) network and local networks [green oval] that connect a number of devices [yellow box] to the *e-box*.” *Id.*



In the external network, there are three different types of e-box-related entities: *e-box* operators [brown box], *e-box* service providers [purple box] and clients [orange box]. *Id.* “[T]he *e-box* operator [brown box] is the owner of the *e-box* network and, as such, is responsible for managing the individual units.” *Id.* “[S]ervice providers [purple box] furnish services to *e-box* end users” in the form of “an *e-box* application

or a piece of software that implements a service.” *Id.* “[A] client [orange box] is any form of terminal equipment that, when connected to the external network, provides a user interface for interacting with a service application on an *e-box*.” *Id.*

The architecture of the edge server (or *e-box* server) installed in the home included hardware corresponding to “a standard PC.” *Id.*, 7. The “system software” included an “operating system,” “drivers for external and local networks,” “server components -- for example, Web servers, wireless application protocol (WAP) servers and domain name servers (DNS),” and “a Java runtime in form of a Java virtual machine (JVM).” *Id.*

The Vasell Provisional Application disclosed that each service application is “basically a Java application, albeit a restricted one.” *Id.*, 7-8. The Vasell Provisional Application explained that “[t]he type of Java application allowed by the service platform is called a *boxlet*, which consequently, is a piece of Java code that implements a service.” *Id.* (emphasis added). The Vasell Provisional Application went on to explain that “[b]oxlets are created using a standard<sup>6</sup> Java development environment.” *Id.*

---

<sup>6</sup> The “standard” Java development environment at the time was described in Section V, *supra*.

**B. U.S. Patent No. 6,496,575 to Vasell (Ex. 1004)**

U.S. Patent No. 6,496,575 (the “Vasell Patent”) (Ex. 1004) incorporated the Vasell Provisional Application by reference. Figure 2 of the Vasell Patent (reproduced below with colored annotations added) is similar to Figure 2 of the Vasell Provisional Application.

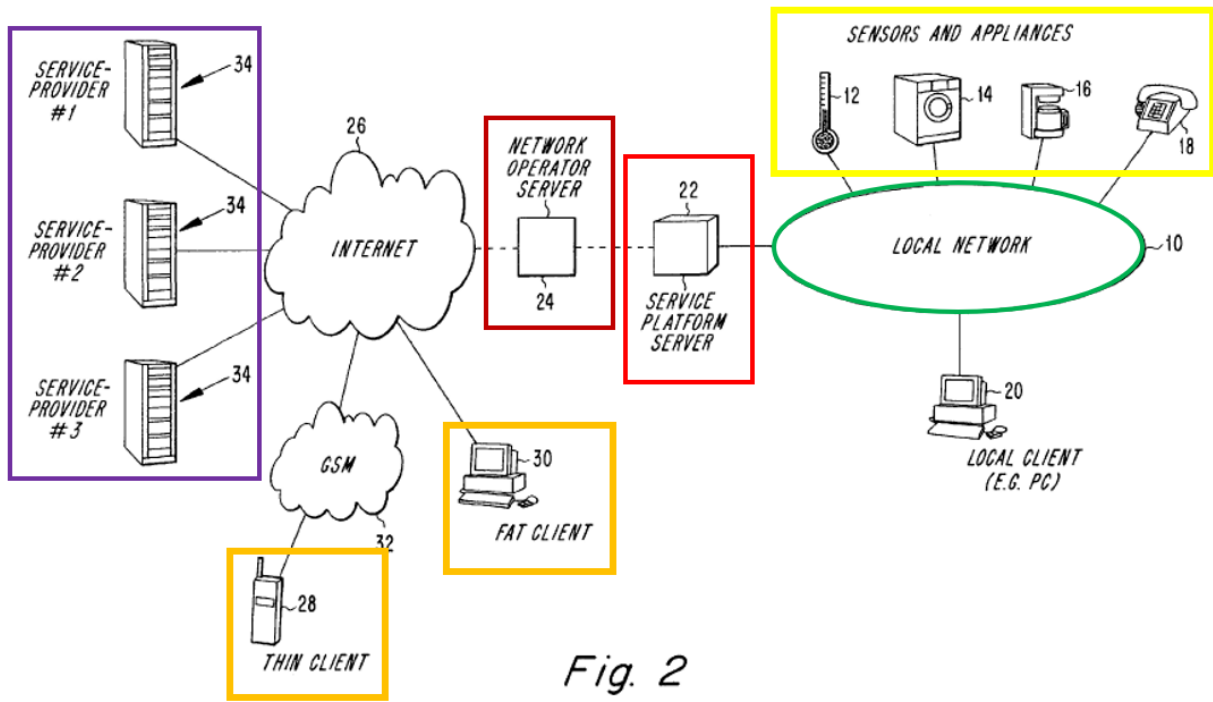


Fig. 2

The platform included a “service platform server 22” (red box) which generally corresponded to the edge server (or *e-box*) disclosed in the Vasell Provisional Application. Ex. 1004, 5:61-6:8, 6:22-44. Service platform server 22 connected to local network 10 [green oval], which connected a number of devices [yellow box] to the service platform server 22. The platform also included a network operator server 24 [brown box] which generally corresponded to one of the “*e-box* operators” in the Vasell Provisional Application, service providers 34 [purple box] which

generally corresponded to the *e-box* service providers in the Vasell Provisional Application, and thin and fat clients [orange boxes] which generally corresponded to the clients in the Vasell Provisional Application.

Figure 6 (reproduced below with colored annotations added) of the Vasell Patent disclosed a main services layer 100 [green box] that included “Cell Manager 93” which starts and stops cells (*id.*, 16:1-4 and 16:29-31), “Cell tables 94” which configure cells and stores the state of service applications (*id.*, 13:60-62 and 21:13-28), a “Security Manager 98” that controls access to classes by cells (*id.*, 15:43-46), and a “Gate Manager 96” which issues gates. *Id.*, 18:56-60. The main services layer 100 also included class libraries 95 which are “essentially code archives” (*id.*, 15:10-14) and class loaders 97 which “load and install” classes. *Id.*, 16:10-12.

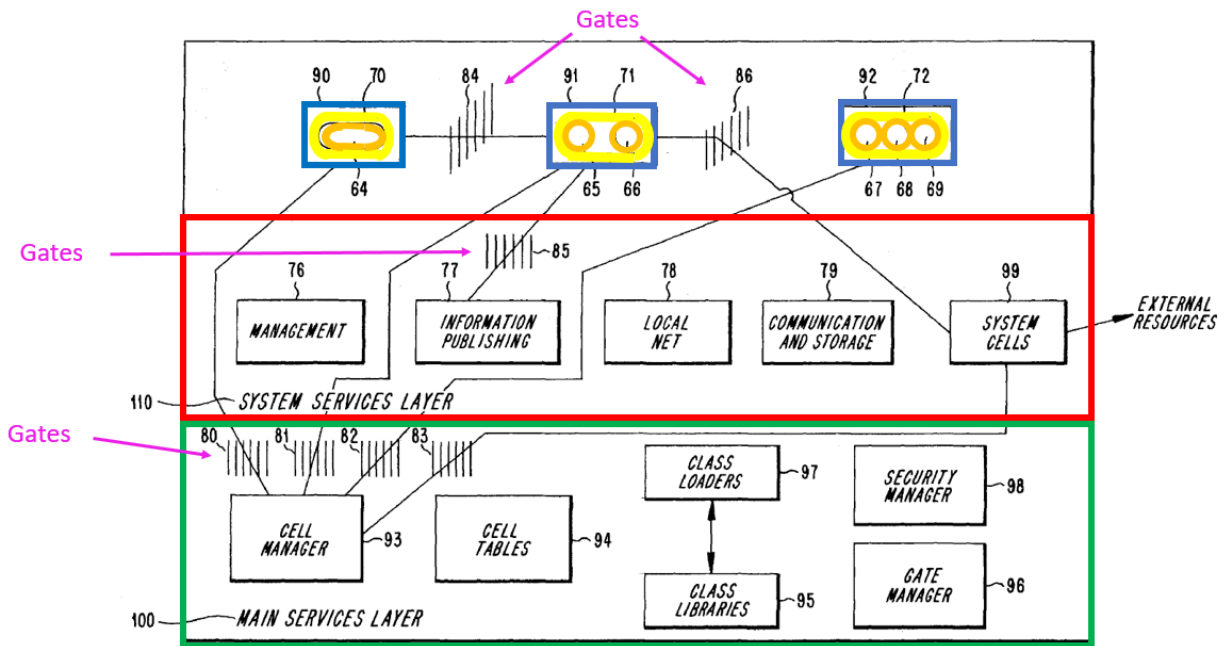


FIG. 6

The service applications 70-72 (yellow ovals) are made up of boxlets 64-69 (orange circles) which implement the service associated with each service application. *Id.*, 13:28-33. The cells 90-92 (blue boxes) represent the resources made available to the service applications 70-72.” *Id.*, 13:50-51 and 14:25-27. “The gates 80-86 [pink arrows] provide and control inter-cell communication. ... The cells 90-92 do not obtain direct references objects in other cells 90-92, but instead communicate to the other cells 90-92 via the gates 80-86. Thus, the gates 80-86 serve as controlled interfaces as the service applications 70-72 within the cells 90-92 interact with each other, with the main services layer 100, or with system services layer 110 [red box].” *Id.*, 18:1-19.

**C. U.S. Provisional Patent Application No. 60/088,437 (Ex. 1015)**

Exhibit 1015 is a provisional patent filing that Vasell purportedly incorporates.

**D. “OSGi in Depth” by Alves (Ex. 1008)**

Alves is directed to the Open Service Gateway Initiative (OSGi), and discloses how “OSGi technology can be used to write better software, and in particular, how OSGi can be used to write better platforms for the development of better software.” Ex. 1008, xviii.

**E. “OSGi in Action - Creating Modular Applications in Java” by Hall (Ex. 1009)**

Hall is also directed to OSGi, and discusses the most common concepts,

features, and mechanisms needed to work with OSGi technology.

**F. “Dependability as a cloud service - a modular approach” by Rellermeyer (Ex. 1011)**

Rellermeyer is directed to “replication of services on distributed cloud platforms.” Ex. 1011, 1.

**VII. A PERSON HAVING ORDINARY SKILL IN THE ART**

For purposes of these proceedings only, Patent Owner does not contest Petitioner’s proposed definition of a POSITA.

**VIII. THE PETITION FAILS TO MEET ITS BURDEN UNDER GROUND 1 TO ESTABLISH THAT THE CHALLENGED CLAIMS WOULD HAVE BEEN OBVIOUS**

In Ground 1, the Petition alleges that claims 1–2, 12, 15, and 19 would have been obvious over Vasell (Exs. 1004, 1015, and 1016) in view of Alves (Ex. 1008) and Rellermeyer (Ex. 1011). Central to Ground 1 is the Petition’s assertion that Vasell discloses or suggests the claim limitation requiring “unified capabilities that enable 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.”

This assertion fails for the reasons explained below.

**A. Ground 1 Fails To Demonstrate That The Relied On References Meet All Of The Limitations Of Claim 1**

This section of the Patent Owner Response begins with a detailed discussion

of the limitations specified in Element [1.4]. Following that discussion, the Patent Owner Response explains (in Sections VIII(A)(2)-(3)) why the relied on prior art fails to meet Element [1.4].

### **1. The Limitations Of Claim 1**

As explained in Section IV, *supra*, 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 runtime*<sup>7</sup> 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. Ex. 2003, ¶98.

In addition, even if “are in secure communication” in Element [1.3] is treated as describing a capability or configuration rather than a presently-running runtime state, Element [1.4] still requires “unified capabilities” enabling “upper layer APIs” to program both the network-device applications and the cloud applications independent of network-device hardware and cloud hardware. That is a distinct requirement that cannot be met by pointing to generic secure channels, generic Java

---

<sup>7</sup> The use of “runtime” is meant to signify that the distributed applications are in a state where they are operating (or running).

development APIs, or generic lifecycle management. Vasell does not disclose any upper-layer API layer that unifies capabilities across the device and cloud domains to program both sides in a hardware-independent way; Petitioner's mapping succeeds only by collapsing these limitations into generic programming and generic management, contrary to the claim language and intrinsic record.

In the next section, the Patent Owner Response explains why the relied on prior art fails to meet the requirements of the claim language discussed above.

## **2. The Petition Fails To Demonstrate That Element [1.4] Is Met By Vasell**

The Petition alleges that the totality of Element [1.4] is disclosed in Vasell alone. However, neither the Vasell Provisional Application nor the Vasell Patent discloses (i) an *upper layer* application programming interface, (ii) which is used to *holistically* program a distributed application *as a whole*, (iii) when the network and cloud applications forming the distributed application are operating or running, and (iv) where the unified interface abstracts away the heterogeneous characteristics of both the network device hardware and the cloud device hardware, as required by Element [1.4]. The Petition citations to Vasell in connection with Element [1.4] are discussed next.

### **a) Vasell's "Write-Once, Run-Everywhere" Maxim And Where Boxlets Interact With The E-Service Infrastructure Through Java Application Program Interfaces Fails To Meet Element 1.4**

First, in connection with Element [1.4], the Petition relies on the following disclosure in Vasell concerning the software development environment:

*Development environment for application software*

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 nonspecialists to develop service applications more easily. Ex. 1016, 4 (emphasis added).

As Dr. Ranka explains, Vasell’s disclosure of a “write-once, run-everywhere” model where boxlets interact with the e-service infrastructure through Java application program interfaces, fails to meet Element [1.4] because, *inter alia*, this disclosure is referring to interfaces at a low layer close to the hardware rather than upper layer interfaces as required by the claim language:

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]. Ex. 2003, ¶110.

In addition, Vasell's disclosure of a "write-once, run-everywhere" model where boxlets interact with the e-service infrastructure through Java application program interfaces fails to disclose "unified capabilities" enabled by the upper layer APIs that provide the user with the capability to program the distributed application *as a whole or holistically*, without an understanding of which portion of Vasell's "distributed application" comprises the network application component of the distributed application and which portion comprises the cloud application component of the distributed application. The Petition maps a boxlet on Vasell's "service platform server 22" (also called the e-box) to the network application component of the distributed application and maps the cloud application component of the distributed application to software on Vasell's "service provider equipment 34" or on Vasell's "network operator equipment 24." Petition, 46-47. Nowhere does Vasell suggest the unified capability to holistically program both application parts of a distributed application (i.e., the part on Vasell's "service platform server 22" and the part on Vasell's "service provider equipment 34" or on Vasell's "network operator equipment 24") with an API that abstracts out the portion of the

distributed application comprising the network application component and the portion comprising the cloud application component and thus allows for the *unified* programming of a distributed application as a whole. Ex. 2003, ¶142. In fact, the subject passage from Vasell speaks at most to programming of software on the e-box (mapped to the claimed network device) and suggests programming on the devices that the Petition maps to the claimed cloud device.

Finally, Vasell’s disclosure of the “Development environment for application software” is referring to APIs “that comply with mainstream Java development,” which were discussed in detail in Section V, *supra*. “Mainstream” APIs are ones used to write the original source code underlying an application itself, rather than to program the application *after* it became operational in a Java Runtime System. Ex. 2003, ¶69. In contrast to the “mainstream” APIs in the Vasell Provisional Application, the specification of the ‘823 Patent describes APIs that can be used “to program” network and cloud applications that are already operating (or running).

As explained in Section V, *supra*, in mainstream Java development, a developer initially wrote Java Language source code (.java files) using APIs associated with pre-written Java classes stored in Java Class Libraries. The developer used a Java Compiler to compile the source code into bytecodes (.class files). A class loader loaded the Java class libraries associated with APIs referenced in the source code, and the bytecodes were loaded into memory and verified before

they entered a Java Virtual Machine. Once in the Java Virtual Machine, the bytecodes were interpreted by either an Interpreter or a just-in-time (JIT) code generator and the application was run in the Runtime System within the Java Virtual Machine. Ex. 2003, ¶68. In this process, the mainstream Java APIs were used as an input to generate the application *before* the application was run. Nothing in “mainstream Java APIs” either expressly or inherently discloses the use of upper layer APIs “*to program*” network and cloud applications that *are* already *operating* (or running), as required by claim 1. Ex. 2003, ¶69.

**b) Vasell’s Disclosures Relating To System Software Fail To Meet Element 1.4**

Next, in connection with Element [1.4], the Petition relies on the following disclosure in the Vasell Provisional Application concerning the “system software” in the e-box:

System software

The system software consists of

- an operating system;
- drivers for external and local network access;
- server components-for example, Web servers, wireless application protocol (WAP) servers and domain name servers (DNS);
- a Java runtime in the form of a Java virtual machine (JVM). Ex. 1016, 7.

Nothing about this above disclosure refers to APIs at all -- let alone upper layer APIs. Moreover, nothing in this disclosure suggests “*unified capabilities*”

enabled by the upper layer APIs that provide the user with the capability to program the distributed application as a whole or holistically, without an understanding of which portion of Vasell’s “distributed application” comprises the network application component of the distributed application and which portion comprises the cloud application component of the distributed application. Finally, the above disclosure is referring to a standard Java Virtual Machine. Again, as explained in the section above, nothing about the existence or use of a standard Java Virtual Machine as set forth in the Vasell Provisional Application discloses (either expressly or inherently) the use of upper layer APIs “to program” network and cloud applications that are already operating (or running), as required by Element [1.4]. Ex. 2003, ¶112-113 and 144-149.

**c) Vasell’s Disclosures Relating To Service Applications And Boxlet Development Fail To Meet Element 1.4**

Next, in connection with Element [1.4], the Petition relies on the disclosures in the Vasell Provisional Application that a service is implemented as a Java application that runs in a Java environment:

*Service applications*

A service application consists of three components:

- A boxlet
- A cell
- One or more gates

The service platform is a Java environment; that is, a service is basically a Java application, albeit a somewhat restricted one. The type of Java application allowed by the service platform is called a *boxlet*, which consequently, is a piece of Java code that implements a service.

\* \* \*

### *Development environment*

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 boxlet development are the libraries that contain APIs for the main services and system services layers. Ex. 1016, 8.

As Dr. Ranka explains, these disclosures fail to meet the requirements of Element [1.4] because, *inter alia*, the APIs are not *upper layer* APIs:

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 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. Ex. 2003, ¶111.

Moreover, nothing about Vasell’s disclosures relating to “Service applications” or its “Development environment” set forth above suggests “unified capabilities” enabled by the upper layer APIs that provide the user with the capability to program the distributed application as a whole or holistically, without

an understanding of which portion of Vasell’s “distributed application” comprises the network application component of the distributed application and which portion comprises the cloud application component of the distributed application. Ex. 2003, ¶144-149.

Finally, Vasell’s disclosures relating to “Service applications” or its “Development environment” set forth above are referring to Java code which, for reasons previously explained, fail to either expressly or inherently disclose the use of upper layer APIs “*to program*” network and cloud applications that *are* already *operating* (or running), as required by claim 1. Ex. 2003, ¶69.

**d) Vasell’s Disclosures Relating To Gates Operating In A Java Virtual Machine Fail To Meet Element 1.4**

The Petition also relies on the following disclosure from the Vasell Patent for Element [1.4]:

The gates 80-86, which each represent an inter-cell connection, are characterized by the following features. The gates 80-86 are each specific to one pair of the cells 90-92 and cannot be handed over to another cell or a third party. Each of the gates 80---86 works locally within a single local operating environment such as a Java Virtual Machine (JVM), or the like. In an alternative embodiment, remote gates may be provided which reside in multiple JVMs. Ex. 1004, 18:23-30.

This disclosure relates to the operating environment of “gates” in Vasell. As explained in Section V above, in Vasell a gate is “[t]he mechanism used for communication between applications.” Ex. 1016, 7-8. The fact that the

communication mechanism *between* application runs in a Java Virtual Machine does not disclose or suggest the use of *upper layer* APIs “*to program*” network and cloud applications that *are* already *operating* (or running), or “*unified capabilities*” enabled by the upper layer APIs that provide the user with the capability to program the distributed application *as a whole or holistically*, without an understanding of which portion of Vasell’s “distributed application” comprises the network application component of the distributed application and which portion comprises the cloud application component of the distributed application.

**e) Vasell’s Disclosures Relating To Its Service Gateway System Fail To Meet Element 1.4**

The Petition also relies on the following disclosures from the Vasell Patent for Element [1.4]: Ex. 1004, 2:60-3:1, 3:10-19, 5:15-35, 10:18-23 and 20:49-54. However, none of these passages even disclose an API, let alone an upper layer API to program distributed applications as recited in Element [1.4]. Ex. 2003, 100-103, ¶¶106, 119-126 and 131-132.

**f) Vasell’s Disclosures Relating To Program Interfaces For The Main Services Layer 100 and The System Services Layer 110 System Fail To Meet Element 1.4**

The Petition also relies on the following disclosures from the Vasell Patent for Element [1.4]:

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.

As Dr. Ranka explains, the “service application 70-72 program interfaces for the main services layer 100 and the system services layer 110” 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, rather than providing upper layer functions for programming distributed applications,” as required by Element [1.4]. Ex. 2003, ¶104 (emphasis added).

Nor do the above disclosures suggest APIs “to program” network and cloud applications that are already operating (or running), or “unified capabilities” enabled by the upper layer APIs that provide the user with the capability to program the distributed application as a whole or holistically, without an understanding of which portion of Vasell’s “distributed application” comprises the network application component of the distributed application and which portion comprises the cloud application component of the distributed application. Ex. 2003, ¶¶128, 146-149.

**g) Vasell’s Disclosures Relating To Its System Management Program Interface Fail To Meet Element 1.4**

The Petition also relies on the following disclosures from Vasell for Element

[1.4]:

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.

As Dr. Ranka explains, 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.” Ex. 2003, ¶105.

Nor do the above disclosures suggest upper layer APIs that provide the user with the capability to program the distributed application as a whole or holistically, without an understanding of which portion of Vasell’s “distributed application” comprises the network application component of the distributed application and which portion comprises the cloud application component of the distributed application, as required by Element [1.4]. Ex. 2003, ¶130. In fact, this disclosure from Vasell (Ex. 1004, 16:55-60) is limited to operations on the e-box (mapped to the “programmable network device”) and has nothing to do with software on

Vasell’s “service provider equipment 34” or on Vasell’s “network operator equipment 24” (mapped to the “programmable cloud device”). As such, this disclosure from Vasell does not suggest functionality to program the “plurality of cloud applications” on the programmable cloud device as required by Element [1.4].

**h) Vasell’s Disclosures Relating To Its Management System Service 76 Fails To Meet Element 1.4**

The Petition also relies on the following disclosures from Vasell for Element [1.4]:

“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.

As Dr. Ranka explains, the “external interface” provided by the management system service 76 “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 required by Element [1.4]. Ex. 2003, ¶107.

Nor do the above disclosures suggest upper layer APIs that provide the user with the capability to program the distributed application *as a whole or holistically*,

without an understanding of which portion of Vasell’s “distributed application” comprises the network application component of the distributed application and which portion comprises the cloud application component of the distributed application, as required by Element [1.4]. Ex. 2003, ¶¶134 and 146-149. In fact, this disclosure from Vasell (Ex. 1004, 22:35-38) is limited to operations on the e-box (mapped to the “programmable network device”) and has nothing to do with software on Vasell’s “service provider equipment 34” or on Vasell’s “network operator equipment 24” (mapped to the “programmable cloud device”). As such, this disclosure from Vasell does not suggest functionality to program the “plurality of cloud applications” on the programmable cloud device as required by Element [1.4].

For these reasons, neither the Vasell Provisional Application or the Vasell Patent discloses (expressly or inherently) the limitations of Element [1.4].

### **3. The “Five Reasons” In The Petition Fail To Demonstrate That Element [1.4] Is Met By Vasell**

The Petition advances “five reasons” why Element [1.4] is met by Vasell.

From the Petition:

POSITAs understood that Vasell discloses this limitation given that (1) the distributed Java service applications are naturally modular; (2) the respective underlying Java software components, of such distributed service applications, use standardized communication protocols to communicate with each other; (3) standard Java APIs are used for their development; (4) “flexible”, “transparent”, and standards-compatible interfaces are used for their lifecycle management in the

service gateway system; and (5) these standard development and management APIs are usable regardless of the respective service platform servers' hardware and remote (cloud) servers' hardware on which the respective JVMs run on the respective OSs. EX1016, 4, 6, 8-9; EX1004, 2:60-3:1, 5:15-35, 22:35-37, 20:52-53, 16:59-60, FIG. 6; EX1003, ¶¶462-463. Petition, 55-56.

Each of the “five reasons” is discussed next. As explained below, these reasons are based in significant part on concepts that are absent from Vasell and which, in any event, fail to meet the requirements of Element [1.4].

**a) Reason #1 - Vasell's Alleged Disclosure Of Java Service Applications Are “Naturally Modular”**

For its first reason, the Petition alleges that Vasell meets Element [1.4] because the distributed Java service applications “are naturally modular.” With respect to service applications being “modular,” Vasell states:

... The infrastructure must be flexible, open and modular, to accommodate a range of communication standards and protocols and to allow individual components rather than entire systems to be replaced as new technology is introduced. Ex. 1016, 4.

\* \* \*

... While sharing the same infrastructure, the service applications can be modularized so that from the perspective of the service providers, a service gateway according to the present invention will appear as if it is dedicated to each service provider's respective service. Ex. 1004, 3:18-23.

These disclosures have *nothing* to do with the use APIs for programming applications, and certainly fail to disclose or suggest the use of upper layer APIs “to program” network and cloud applications that *are* already *operating* (or running), as

required by Element [1.4].

**b) Reason #2 - Vasell's Disclosure Of Standardized Communication Protocols**

For its second reason, the Petition alleges that Vasell meets Element [1.4] because “the respective underlying Java software components, of such distributed service applications, use standardized communication protocols to communicate with each other.” The fact that the underlying Java software components “use standardized protocols to communicate with each other” is not a disclosure that APIs are used for programming applications, and certainly fails to disclose or suggest the use of upper layer APIs “to program” network and cloud applications that *are* already *operating* (or running), as required by Element [1.4].

**c) Reason #3 - Vasell's Disclosure Of Standard Java APIs**

Petitioner's Element [1.4] theory improperly equates the claimed “plurality of upper layer application programming interfaces (APIs)” with general-purpose programming-language development APIs (e.g., standard Java APIs and associated libraries) used to author and compile application source code, rather than configuring an application that is executing. That is a category error. The '823 Patent uses “upper layers” in a specific architectural sense tied to its platform stack: it discloses “tools,” “native daemons,” “native libraries,” and “Hardware Abstraction Layers (HAL),” and explains that these components “abstract out the hardware dependencies for the upper layers and programmers.” EX1001, 12:17–22. Thus, “upper layer” refers to

platform layers positioned above the HAL—i.e., the layer(s) explicitly insulated from hardware-specific dependencies by the platform’s abstraction mechanisms. Ex. 1001, 12:17–22.

The intrinsic record reinforces this runtime understanding. The ‘823 Patent describes a secure connection that “allows communication between fxDeviceApps . . . and fxCloudApp,” and explains that fxOS controls application access to platform APIs, including enforcing authorization limits based on application access levels. EX1001, 11:50–56. That description presupposes running applications that can communicate and that can invoke and respond to platform APIs under fxOS control. Similarly, the ‘823 Patent explains that fxCloud and fxOS “create a virtual fabric (fxVF) for messaging between applications,” and that messaging complexity can be abstracted for developers. EX1001, 11:56–60. Messaging between applications and platform-controlled API access are runtime behaviors; they occur when applications are active and capable of receiving and processing API calls. These disclosures confirm that the “upper layer APIs” in Element [1.4] are contemplated as interfaces used to program and control operational application behavior within the distributed environment, not merely tools used to author code or to manage static software artifacts. EX1001, 11:50–60; 12:17–22; 12:26–39.

Accordingly, Petitioner’s reliance on generic “lifecycle management” and

general-purpose development APIs is insufficient to meet the claim language. At most, installing or downloading an application modifies stored software state. It does not establish that a plurality of upper-layer APIs “program” the plurality of applications in the sense required by Element [1.4], because offline lifecycle operations do not involve the applications themselves receiving and responding to API calls as active components in the distributed system. Put differently, Petitioner’s theory attempts to treat “program” as satisfied by activities that can occur when the application is not executing, collapses Element [1.4] into generic deployment and administrative management, and ignores the claimed API-driven runtime programming. EX1001, 11:50–60; 12:26–39; 12:48–54.

As a result, Petitioner cannot satisfy Element [1.4] by pointing to generic programming-language APIs (e.g., “standard Java APIs”) that merely facilitate authoring portable code, or by pointing to generic lifecycle operations (download/install/remove/execute) that do not constitute the claimed “unified capabilities” enabling “upper layer APIs” to program both the network-device applications and the cloud applications independent of both hardware domains. Rather, Element [1.4] requires that the applications being “program[med]” by the claimed APIs be running (or at minimum active in a runtime sense such that they can receive and respond to API invocations). Because Petitioner’s asserted prior art does not disclose APIs that program both the network-device applications and the

cloud applications as active components of distributed applications, Petitioner fails to meet its burden for Element [1.4].

**d) Reason #4 - Vasell's Alleged Disclosure Of Flexible, Transparent, And Standards-Compatible Interfaces For Lifecycle management**

For its fourth reason, the Petition alleges that Vasell meets Element [1.4] because “‘flexible’, ‘transparent’, and standards-compatible interfaces are used for ... lifecycle management.” This reason is flawed on many levels. Starting with the term “flexible,” with respect to this term, Vasell discloses only the following:

... it is a purpose of the present invention to provide secure, robust connectivity based service gateway or services platforms. It is a further purpose of the present invention to provide a *flexible system* for remotely managing the service gateways, while maintaining compatibility with existing information and communication technologies, protocols and interface standards. Ex. 1004, 2:44-50 (emphasis added).

\* \* \*

The *infrastructure* must be *flexible*, open and modular, to accommodate a range of communication standards and protocols and to allow individual components rather than entire systems to be replaced as new technology is introduced. Ex. 1016, 4 (emphasis added).

These disclosures do not say that Vasell's *interfaces* will be flexible. Rather, they disclose a need for flexibility in other aspects of the system so that the platform can accommodate a range of existing interface standards. The passages are teaching the opposite of changing existing interfaces to make them flexible. Instead, these

passages are teaching that existing interfaces will not change so *the rest of the system* must be *flexible* to accommodate those interfaces.

Turning next to the term “lifecycle,” with respect to this term, Vasell discloses only the following:

On the *e-box* service platform, applications that implement services can be downloaded, installed or removed. The platform also permits the remote *life-cycle management* of service applications. Ex. 1016, 7 (emphasis added).

This passage does not disclose that “standards-compatible interfaces are used for ... lifecycle management” in Vasell.

Finally, even if Vasell disclosed that “‘flexible’, ‘transparent’, and standards-compatible interfaces are used for ... lifecycle management,” such a disclosure still would not satisfy the claim language of Element [1.4] which requires *upper layer* APIs “to program” distributed applications that *are* already *operating* (or running), including a “*unified capability*” through which the user programs the distributed application holistically or as a whole.

Accordingly, even if Petitioner were correct that lifecycle-management operations can be characterized as “programming” in some abstract sense, that does not carry Petitioner’s burden under Element [1.4]. Element [1.4] requires more than (i) an interface to download/install/remove software or (ii) a general-purpose programming environment used to author software. It requires “*unified capabilities*”

supported by “upper layer APIs” that program both network-device applications and cloud applications that form distributed application in a hardware-independent way. Under this intrinsic, architecture-driven understanding of “upper layer APIs” and “unified capabilities,” a prior-art disclosure focused on generic lifecycle management (even if secure) is not sufficient, because it does not disclose a unified, upper-layer programming interface that programs both sides of distributed applications independent of both hardware domains. EX1001, 12:17–22; 12:26–33.

e) **Reason #5 - Vasell’s Alleged Disclosure Of Standard Development And Management APIs Are Usable Regardless Of The Respective Service Platform Servers’ Hardware And Remote (Cloud) Servers’ Hardware On Which The Respective JVMs Run On The Respective OSs**

For its fifth reason, the Petition alleges that Vasell meets Element [1.4] because it discloses that “standard development and management APIs are usable regardless of the respective service platform servers’ hardware and remote (cloud) servers’ hardware on which the respective JVMs run on the respective OSs.” Even if Vasell disclosed such management APIs, such a disclosure still would not satisfy Element [1.4] which requires upper layer APIs “to program” distributed applications that are already operating (or running), including a “unified capability” through which the user programs the distributed application holistically or as a whole.

**4. Vasell’s “External Interface” Does Not Expressly Or Inherently Disclose A Plurality Of Application Programming Interfaces For Programming The Plurality Of Network Device And Cloud Applications**

The Vasell Patent includes the following disclosure with respect to service applications:

The system services 76-79 of the system services layer 110 includes a management system service 76, an information publishing system service 77, a local net system service 35 78, and a communication and storage system service 79. 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 (emphasis added).

Citing this passage, the Institution Decision stated:

Patent Owner also argues that the APIs used for the development of boxlets are “for ‘main services and system layers’ and not for programming applications.” Prelim. Resp 26. This argument is inconsistent with Vasell’s description which states that the APIs for the main services and system services layers are “specific to boxlet development,” indicating that they are, in fact, for programming the service applications. ... [Ex. 1004], 22:35–38 (explaining that the system service layer includes a management system service 76 that downloads, installs, removes, executes, and controls service applications). To the extent Patent Owner is arguing that the claim term “program” excludes the management, installation, removal, execution, and control of service applications, we disagree based on the current record. Institution Decision, 19-20 (emphasis added).

In response, Patent Owner wishes to make clear its position with respect to the issue identified by the Board. As explained in Section VIII(A)(A)(h), the “external interface” referenced in Vasell (at 22:35-38) provided by the management system

service 76 “functions as a *lower layer* management API, rather than providing upper layer APIs for programming network and cloud applications that form distributed applications,” as required by Element [1.4]. Ex. 2003, ¶¶107. Nor do the disclosures in Vasell (at 22:35-38) suggest upper layer APIs that provide the user with the capability to program a distributed application *as a whole or holistically*, without an understanding of which portion of Vasell’s “distributed application” comprises the network application component of the distributed application and which portion comprises the cloud application component of the distributed application, as required by Element [1.4]. Ex. 2003, ¶, 134, 146-149. In fact, the cited disclosure from Vasell (Ex. 1004, 22:35-38) is limited to operations on the e-box (mapped to the “programmable network device”) and has nothing to do with software on Vasell’s “service provider equipment 34” or on Vasell’s “network operator equipment 24” (mapped to the “programmable cloud device”).

Even under a capability or configuration reading of Element [1.3], Petitioner still must show that the “unified capabilities” and “upper layer APIs” of Element [1.4] program *both* the plurality of network-device applications and the plurality of cloud applications, independent of both hardware domains. Vasell’s “remote life-cycle management” disclosure, at most, concerns management of service applications on the e-box platform. It does not disclose any plurality of upper-layer APIs that program the applications Petitioner maps to the network operator server

24 or service provider servers 34 (mapped to the claimed cloud device), and Petitioner nowhere identifies an “external interface” that is integrated into, or that programs, those server-side application components.

For example, the Petition maps “the plurality of network device applications” to the “portions/components” of Vasell’s “distributed” service applications “hosted on server platform server 22.” Petition, 46-47. The Petition then maps the “plurality of cloud applications” to the “portions/components” of Vasell’s “distributed” service applications hosted on either network operator server 24 or server provider servers 34. Petition, 47-49. These mappings are depicted in the following annotated diagram from the Petition, which uses orange to depict “a cloud application that is a component of the distributed application hosted on a cloud device” and pink to depict a “network device application that is a component of the distributed application hosted on a network device.”<sup>8</sup>

---

<sup>8</sup> This color coding scheme is set forth in the Petition, at 1-2.

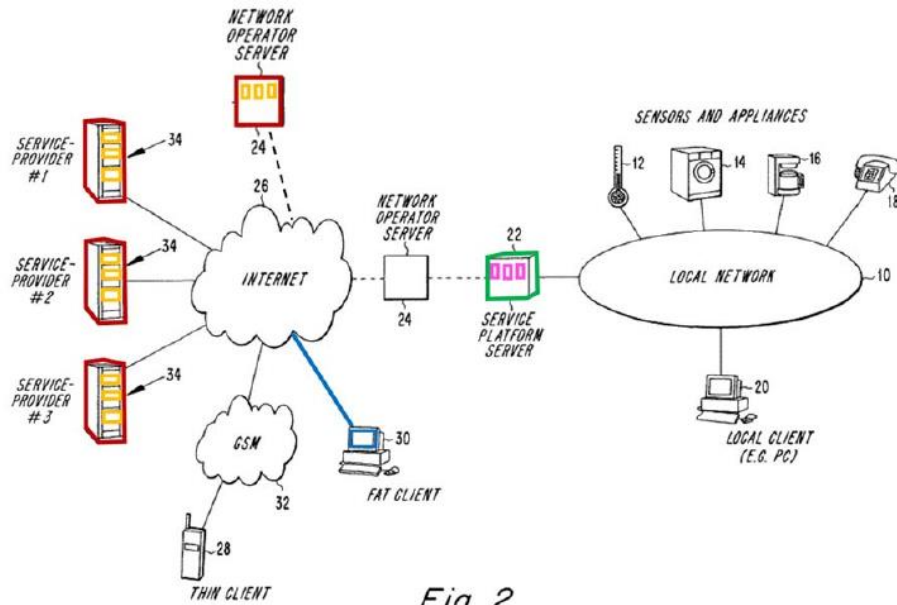


Fig. 2

Significantly, Vasell’s “external interface” is disclosed as being integrated with Vasell’s “management system service 76” located in the System Services Layer of 110 of Vasell’s server platform server 22 (which the Petition alleges hosts “network device applications”). Ex. 1004, Figure 6, 21:33-40 and 22:19-21.

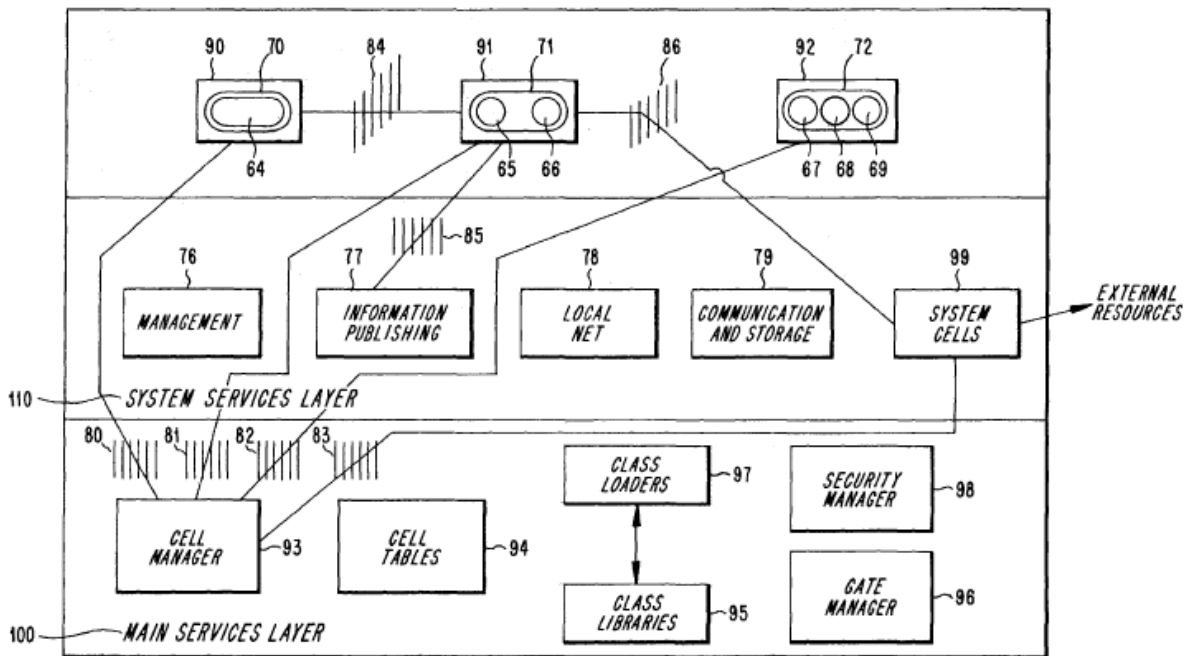


FIG. 6

Vasell does not disclose that the “external interface” is integrated into either network operator server 24 or server provider servers 34 (which the Petition alleges host the “cloud applications”). Significantly, the plurality of upper layer APIs recited in Element [1.4] must be capable of programming “the plurality of network device application and plurality of cloud applications.” Simply put, since Vasell’s “external interface” is not integrated into either network operator server 24 or server provider servers 34 (which the Petition alleges host the “cloud applications”), the “external interface” clearly lacks any capability of programming the “cloud applications” allegedly hosted on those devices.

For the above reasons, the subject passage in Vasell (at 22:32-38) that refers

to an “external interface” fails to expressly or inherently disclose “a plurality of upper layer APIs” with the capability “to program the plurality of network device applications and plurality of cloud applications,” as required by Element [1.4].

**B. Petitioner Fails to Establish a Motivation to Combine or a Reasonable Expectation of Success**

Petitioner bears the burden to prove that a POSITA would have been motivated to combine the cited references and would have had a reasonable expectation of success in doing so. Petitioner does not satisfy that burden here. Instead, Petitioner’s “motivation” narrative largely amounts to the assertion that Alves, Rellermeyer, and Hall provide “implementation details” that a POSITA would have applied to Vasell. That generic assertion is insufficient where Petitioner’s theory only works by re-architecting Vasell into a fundamentally different platform that purportedly provides “unified capabilities” and “upper layer APIs” to program distributed applications across both device-side and cloud-side components independent of both hardware domains. Element [1.4] is not satisfied by simply implementing Vasell using an OSGi framework, adding generic cloud dependability concepts, or incorporating signatures. Petitioner’s rationale assumes the claimed objective of unified, upper-layer, cross-domain programmability, and then retrofits the prior art to reach it.

**1. Petitioner’s proposed combination requires wholesale redesign, not routine “implementation details”**

Petitioner’s theory is not a straightforward substitution of known components. To reach Element [1.4], Petitioner must transform Vasell’s system into a platform in which a set of “upper layer APIs” provides “unified capabilities” to program both the network-device applications and the cloud applications independent of both network-device hardware and cloud hardware. That is not a narrow optimization or an engineering detail. It is a platform-level redesign that changes how the system is structured, how functionality is exposed to developers, and how device-side and cloud-side components are programmed and coordinated.

Nothing in Vasell suggests that its remote management concepts, or its desire for compatibility with existing protocols and interface standards, would motivate a POSITA to replace Vasell’s approach with a new, unified, upper-layer programming framework spanning both device and cloud domains. To the contrary, Vasell describes maintaining compatibility with existing protocols and interface standards while providing flexibility in the overall system. Ex. 1004, 2:44–50. That is not a teaching to create a new unified API abstraction layer spanning heterogeneous device and cloud environments. It is a teaching to accommodate existing standards and interfaces without requiring that the interfaces be changed.

- 2. The combination is driven by hindsight because it presupposes the claimed “unified capabilities” objective**

Petitioner's motivation theory begins at the endpoint. It assumes that the correct goal is to provide unified, upper-layer, hardware-independent programming capabilities for distributed applications spanning device and cloud, and then assembles selected features from disparate references to approximate that goal. *KSR* does not permit combining references using the challenged patent as a roadmap. Petitioner must identify a reason grounded in the prior art itself that would have led a POSITA to pursue Petitioner's specific architecture, namely a unified, upper-layer API layer that programs both sides of the distributed application independent of both hardware domains.

Petitioner does not do so. Instead, Petitioner relies on generalized statements that OSGi provides modularity or that cloud systems benefit from dependability and scalability, and then treats those concepts as if they supply the motivation to rework Vasell into a cross-domain unified programming platform. That is hindsight. The "reason to combine" comes from the claim limitations themselves, not from an articulated problem in Vasell that the proposed redesign would have been undertaken to solve.

**3. Petitioner also fails to prove a reasonable expectation of success for its re-architecture**

Even if a POSITA had a reason to add modularity, repositories, or dependability concepts, Petitioner still must show a reasonable expectation of

success in achieving the claimed result. Petitioner does not. The asserted references address different problems at different layers. OSGi-related references address modular component packaging and management within a Java or OSGi environment. Cloud dependability references address migration, failover, and load-balancing characteristics. Signature and certificate concepts address authenticity. None of these, alone or together, teaches how to implement the claimed upper-layer, unified programming capabilities that program both device-side and cloud-side application components independent of both hardware domains.

Nor does Petitioner identify any concrete implementation pathway showing how its combination would produce unified capabilities exposed through upper-layer APIs that program both sides of the distributed application across device and cloud while remaining hardware-independent across both environments. Instead, Petitioner treats these disparate teachings as interchangeable “details” and assumes they can be stitched into Vasell without material friction. That assumption is unsupported and does not satisfy Petitioner’s burden.

**C. Ground 1 Fails To Demonstrate That The Relied On References Meet All Of The Limitations Of Claims 2, 12-15 and 19**

Claims 2 and 12-15 depend from claim 1. In Ground 1, the mappings for these dependent claims is predicated on the mapping for claim 1 which, as discussed above, fails to satisfy the claim language. Thus Ground 1 also fails to demonstrate the unpatentability of dependent claims 2 and 12-15.

Ground 1 also challenges independent claim 19. Element [19.4] is substantially the same as Element [1.4], and for Element [19.4] the Petition relies on its previous mapping of Element [1.4] which, as discussed above, fails to satisfy the claim language [19.4]. Petition, 64. Thus, Ground 1 also fails to demonstrate the unpatentability of claim 19.

#### **IX. THE PETITION FAILS TO MEET ITS BURDEN UNDER GROUND 2**

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.

#### **X. CONCLUSION**

Patent Owner respectfully requests that the Board enter a Final Written Decision finding that the Challenged Claims are not unpatentable.

Respectfully submitted,

Dated: December 23, 2025

By: /Brandon R. Theiss/  
Brandon R. Theiss  
Registration No. 70,507

*Counsel for Patent Owner*  
*Edge Networking Systems, LLC*

**CERTIFICATE OF COMPLIANCE WITH WORD COUNT**

Pursuant to 37 C.F.R. § 42.24(d), I certify that this **PATENT OWNER’S RESPONSE** complies with the type-volume limits set forth in the Memorandum and 37 C.F.R. § 42.24 because it contains 13, 737 words, excluding the parts of the Brief that are exempted by 37 C.F.R. § 42.24(a), according to the word processing system used to prepare this Brief.

Respectfully submitted,

Dated: December 23, 2025

By: /Brandon R. Theiss/  
Brandon R. Theiss  
Registration No. 70,507

*Counsel for Patent Owner  
Edge Networking Systems, LLC*

**CERTIFICATE OF SERVICE**

The undersigned certifies that pursuant to 37 C.F.R. § 42.6(e), a copy of the foregoing **PATENT OWNER'S RESPONSE** was served via email (as consented to by counsel) on December 23, 2025 to lead and backup counsel of record for Petitioner as follows:

Christopher J. Tyson  
DUANE MORRIS LLP  
901 New York Avenue N.W., Suite 700 East, Washington, D.C. 20001  
CJTyson@duanemorris.com

John M. Baird  
DUANE MORRIS LLP  
901 New York Avenue N.W., Suite 700 East, Washington, D.C. 20001  
JMBaird@duanemorris.com

Glenn D. Richeson  
DUANE MORRIS LLP  
1075 Peachtree Street NE, Suite 1700, Atlanta, GA 30309  
GDRicheson@duanemorris.com

Patrick D. McPherson  
DUANE MORRIS LLP  
901 New York Avenue N.W., Suite 700 East, Washington, D.C. 20001  
PDMcPherson@duanemorris.com

Respectfully submitted,

Dated: December 23, 2025

By: /Brandon R. Theiss/  
Brandon R. Theiss  
Registration No. 70,507

*Counsel for Patent Owner  
Edge Networking Systems, LLC*