

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

INTERNATIONAL BUSINESS MACHINES CORP.,

Petitioner,

v.

VIRTAMOVE, CORP.,

Patent Owner.

Case IPR2025-00599

U.S. Patent No. 7,519,814

**DECLARATION OF TODD C. MOWRY, PH.D.
IN SUPPORT OF PETITION FOR *INTER PARTES* REVIEW
OF U.S. PATENT NO. 7,519,814**

TABLE OF CONTENTS

I.	Introduction.....	1
II.	Background and Qualifications	2
III.	Documents and Materials Considered.....	6
IV.	Legal Principles	7
V.	Person of Ordinary Skill in the Art.....	13
VI.	Background of the '814 Patent	14
	A. Overview of the '814 Patent.....	15
	B. Prosecution History of the '814 Patent	16
	C. Priority Date of the Challenged Claims	16
VII.	Background of the Prior Art	17
	A. McMillan	17
	B. Gaines	23
	C. Schaefer	20
	D. Tormasov	23
VIII.	Claim Construction.....	27
IX.	Invalidity Opinions: '814 Patent.....	29
	A. Ground 1: McMillan Alone or In View of Gaines and Tormasov.....	30
	1. Claim 1	30
	2. Claim 2: “A method as defined in claim 1, wherein each container has an execution file associated therewith for starting the one or more applications.”	57
	3. Claim 6: “A method as defined in claim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.”	58
	4. Claim 9: “A method as defined in claim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applicatins within the containers being executed.”	65
	5. Claim 10: “A method as defined in claim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating systemduring execution thereof.”	68
	B. Ground II: Schaefer Alone or in View of Gaines and/or Tormasov...	69

1.	Claim 1	69
2.	Claim 2: “A method as defined in claim 1, wherein each container has an execution file associated therewith for starting the one or more applications.”	103
3.	Claim 6: “A method as defined in claim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.”	104
4.	Claim 9: “A method as defined in claim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applicatins within the containers being executed.”	110
5.	Claim 10: “A method as defined in claim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating systemduring execution thereof.”	113
X.	Secondary Considerations Of Non-Obviousness	116

I, Dr. Todd C. Mowry, declare as follows:

I. INTRODUCTION

1. My name is Todd C. Mowry.

2. I have been retained as an expert witness on behalf of International Business Machines Corp. (“IBM” or “Petitioner”) for the above-captioned Petition for inter partes review (the “Petition”) of U.S. Patent No. 7,519,814 (the “’814 Patent”).

3. I understand that IBM is challenging the validity of claims 1, 2, 6, 9, and 10 (the “Challenged Claims”) of the ’814 Patent in the Petition.

4. I have been asked to consider whether certain references invalidate the Challenged Claims.

5. I am being compensated for my time in connection with the Petition at my standard consulting rate of \$650 per hour. My compensation is not affected by the content of my opinions, my testimony, or the outcome of the Petition.

6. I am not currently, and have not at any time in the past been, an employee of IBM. I have no financial interest in IBM.

7. The prior art references on which my opinions are based are:

- U.S. Patent No. 7,673,308 (“McMillan”)
- U.S. Patent No. 5,961,582 (“Gaines”)
- U.S. Patent Pub. No. 2002/0174215 A1 (“Schaefer”)
- U.S. Patent Pub. No. 2002/0124072 A1 (“Tormasov”)

8. In my opinion, the Challenged Claims are invalid based on the following:

- McMillan alone, or alternatively, in combination with Gaines and/or Tormasov.
- Schaefer alone, or alternatively, in combination with Gaines and/or Tormasov.

II. BACKGROUND AND QUALIFICATIONS

9. I am a Professor in the Department of Computer Science at Carnegie Mellon University. I also have a courtesy appointment in the Department of Electrical and Computer Engineering. I have served on the faculty of Carnegie Mellon University for twenty-eight years from 1997 through the present (2025).

10. I also served on the faculty of the University of Toronto for four years between 1993 and 1997, in the Department of Electrical and Computer Engineering and with a courtesy appointment in the Department of Computer Science. Prior to that appointment, I served as a Graduate Research Assistant in the Department of Electrical Engineering at Stanford University for four years between 1989 and 1993.

11. Over the years, I have taught and continue to teach many courses and directed research in computer systems and software, operating systems, distributed and network systems, parallel processing, and compiler optimization.

12. I received a B.S. in Electrical Engineering with Highest Distinction from the University of Virginia in May 1988. I received an M.S. in Electrical

Engineering from Stanford University in June 1989, and a Ph.D. in Electrical Engineering from Stanford University in March 1994.

13. I have worked in the computer industry in various capacities over the years. I was a part-time Computer Architect and then Computer Architecture Consultant at Silicon Graphics, Inc. in Mountain View, California (formerly MIPS Computer Systems in Sunnyvale, California) from 1989 to 1993 and 1993 to 1996, respectively. I was a Visiting Scientist at IBM in Toronto from 1996 to 2004. During that same time period (1996 to 2004), I was also a Member of the Technical Advisory Board of SandCraft, Inc. in Santa Clara, California. I was the Director of the Intel Research Pittsburgh Lab at Intel Corporation in Pittsburgh, Pennsylvania from 2004 to 2007.

14. I have authored twenty-one (21) journal articles and seventy-two (72) conference papers, most of which were peer reviewed. I am also a named inventor on five patents.

15. I have published a number of papers in the top research conferences in operating systems and data storage (including a paper that won the Best Paper Award at the USENIX 2nd Symposium on Operating Systems Design and Implementation (OSDI '96)). *See, e.g.,* Olatunji Ruwase, Michael A. Kozuch, Phillip B. Gibbons, and Todd C. Mowry, *Guardrail: High Fidelity Correctness Checking of Device Drivers for Safeguarding I/O Operations*, in Proceedings of the Nineteenth

International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) (2014); Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd Mowry, Matthew Perron, Ian Quah, Siddharth Santurkar, Anthony Tomasic, Skye Toor, Dana Van Aken, Ziqi Wang, Yingjun Wu, Ran Xian, and Tieying Zhang, *Self-Driving Database Management Systems*, in Proceedings of the 2017 Conference on Innovative Data Systems Research (CIDR) (2017); and Amit Manjhi, Charles Garrod, Bruce M. Maggs, Todd C. Mowry, Anthony Tomasic, *Holistic Query Transformations for Dynamic Web Applications*, in 2009 IEEE 25th International Conference on Data Engineering (ICDE) (2009).

16. I have also been an active member of the Parallel Data Lab at Carnegie Mellon University since 1997, which is often described as “academia’s premiere storage systems research center.”¹

17. I am the recipient of several honors and awards: the Arthur Samuel Thesis Award (awarded by the Stanford Computer Science department to the top two Ph.D. theses in a given year), several IBM Faculty Development Awards (1996, 1997, 1998, 2000, 2001, 2002, and 2003), several Best Paper Awards (the Second Symposium on Operating Systems Design and Implementation in 1996; the 20th

¹ See, e.g., <https://www.pdl.cmu.edu/PDL-FTP/News/newsletter02.pdf>.

International Conference on Data Engineering (ICDE) in 2004), the Alfred P. Sloan Research Fellowship (awarded to researchers in recognition of distinguished performance and a unique potential to make substantial contributions to their field), the Most Thought-Provoking Idea Award in 2004 (awarded by the Architectural Support for Programming Languages and Operating Systems (ASPLOS)), the TR100 Award in 1999 (awarded by MIT's Technology Review magazine to the top 100 most promising young innovators in science and technology), and I became an ACM Fellow in 2016.

18. I am a member of the Institute of Electrical and Electronics Engineers (IEEE) and the Association of Computing Machinery (ACM). I was the Editor-in-Chief of ACM Transactions on Computer Systems from 2013 through 2018, which is the premier journal for computer systems research. (I was an Associate Editor for the journal prior to that, since 2001.) I was the Program Chair of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) in 2010. I was the Co-Program Chair of the International Conference on Parallel Architectures and Compilation Techniques (PACT) in 2001. I have been on the programming committee in various years for ASPLOS, the International Symposium on Computer Architecture (ISCA), the International Symposium on Microarchitectures, and the Workshop on Architectural and System Support for Improving Format.

19. Overall, I have over thirty years of experience in the field of computer science.

20. My qualifications and publications are set forth more fully in my curriculum vitae, attached as Ex-1004.

III. DOCUMENTS AND MATERIALS CONSIDERED

21. In forming the opinions expressed in this Declaration, I relied upon my education and experience in the relevant field of the art and have considered the viewpoint of a person having ordinary skill in the art (POSITA) at the time of the alleged invention.

22. I have considered the materials referenced herein, including the '814 Patent (Ex-1001), the file history of the '814 Patent (Ex-1002), the Petition, and other documents listed in the Exhibit List of the Petition, including:

Description	Date of Public Availability
U.S. Patent No. 7,673,308 ("McMillan") (Ex-1005)	Provisional application filed on November 18, 2002, and patent issued on March 2, 2010. <i>See</i> Ex-1006.
U.S. Provisional Application No. 60/427,339 (the "'339 Application") (Ex-1006)	Filed on November 18, 2002.
U.S. Patent No. 5,961,582 ("Gaines") (Ex-1007)	Filed on October 25, 1994, and issued on October 5, 1999
U.S. Patent Pub. No. 2002/0174215 A1 ("Schaefer") (Ex-1008)	Filed on May 16, 2001, and published on November 21, 2002
Andrew S. Tannenbaum, <i>Modern Operating Systems</i> (2d ed. 2001)	Published in 2001.

(Ex-1009)	
D.A. Solomon, "The Windows NT kernel architecture," in <i>Computer</i> , Vol. 31, No. 10, pp. 40-47, Oct. 1998 (Ex-1010)	Published in 1998.
U.S. Patent Pub. No. 2002/0124072 A1 ("Tormasov") (Ex-1011)	Filed on July 30, 2001 and published on September 5, 2002

23. I have reviewed both U.S. Patent No. 7,673,308 ("McMillan") and its Provisional Application No. 60/427,339 ("McMillan's Provisional Application" OR "'339 Application"), to which it claims priority. The '339 Application's specification is substantially identical to McMillan's specification as issued. See Ex-1014. The only notable additions to McMillan's specification include three new headers: "Reference to Related Application," "Field of the Invention," and "Background of the Invention." No new teachings were added to McMillan's specification. Indeed, the "Background of the Invention" section, which is the most significant addition, is limited to a discussion of what a POSITA would have already known at the time, including the well-known techniques of virtual operating systems and virtual environments. Therefore, while I cite to McMillan's specification for purposes of this Declaration, the same disclosures can be found in the '339 Application. See *id.* Below is a table that provides exemplary citations to the '339 Application directly corresponding to the disclosures found in McMillan, demonstrating that the teachings of McMillan can be found identically in the '339 Application and that McMillan's claim 1 is clearly supported by the '339

Application. *See also* Ex-1014.

McMillan Claim 1 Element	Exemplary McMillan Provisional Disclosure (Ex-1006)	Corresponding McMillan Patent Disclosure
A computing machine having a computing architecture, comprising:	Ex-1006 at 1, Figs. 1-4 (“This invention is directed to the creation of multiple semi-independent virtual OS environments within a single operating system (OS)”).	Ex-1005, at Figs. 1-4, 1:12-14, 2:10-12, 2:55-57.
a base operating system (OS) installed to the computing machine, the base OS having a base OS file system and a base OS registry;	Ex-1006 at 1, Figs. 1-2 (“This invention is directed to the creation of multiple semi-independent virtual OS environments within <i>a single operating system (OS)</i> The file system and registry information for each environment is independent of the <i>base OS</i> and other environments. . . . In addition, all of the applications run on a <i>single OS desktop</i>”) <i>See also</i> at 2–3.	Ex-1005, at Figs. 1-2, 1:12-14, 2:10-19, 2:55-64, 3:1-21, 3:57-59.
at least one virtual OS environment within the base OS, the virtual OS environment having a virtual OS file system and a virtual OS registry which are independent of the base OS file system and the base OS registry, wherein a dynamic link library (DLL) is injected into an application running under the virtual OS environment, the	Ex-1006 at 1, Figs. 1-2 (“This invention is directed to the creation of <i>multiple semi-independent virtual OS environments</i> within a single operating system (OS). . . . In this way <i>each virtual OS environment appears to be an independent OS for the applications running within it. The file system and registry information for each environment is independent of the base OS and other environments</i>”), 2 (“To create <i>the virtual OS environments</i> , the OS APIs that access <i>the file system and registry</i> directly and indirectly must be changed to redirect these accesses into <i>the virtual OS environments file system and registry</i> . <i>This is done by injecting a DLL into</i>	Ex-1005, at Figs. 1-2, 1:12-14, 2:10-19, 2:55-64, 3:1-21, 3:49-4:32, 4:57-65.

DLL being programmed to	<i>every application that is executed.”); see also id. at 3–4.</i>	
determine that the application should be run under the virtual OS environment instead of the base OS;	Ex-1006 at 3, Fig. 2 (“As mentioned earlier, not all applications are run from virtual OS environments and different applications may be run from different virtual OS environments. <i>When the injected DLL is first loaded it must determine which of the virtual OS environments (if any) to redirect to. . . . The base OS contains a list of directories where applications should be redirected to specific virtual OS environments.”); see also id. at 4, 6.</i>	Ex-1005, at Fig. 2, 3:17-21, 3:49-4:32, 4:57-65.
call a function of the injected DLL rather than calling an API of the base OS;	Ex-1006 at 2, Figs. 3-7 (“This is done by injecting a DLL into every application that is executed. . . . When the application (or one of the applications DLLs) attempts to call an OS API (directly or indirectly) that will access the file system or registry, <i>the injected DLL’s function is called instead.”); see also id. at 3–7.</i>	Ex-1005, at Figs. 3-7, 3:17-37, 3:49-5:8, 5:20-37.
modify at least one parameter from a calling function of the application to direct the at least one parameter to a location of the virtual OS environment;	Ex-1006 at 2, Figs. 3-7(“The injected DLL’s function then examines <i>the parameters that are passed to the API and modifies them</i> to direct them into the virtual OS environment location instead.”); see also id. at 3–7.	Ex-1005, at Figs. 3-7, 3:34-39, 3:49-4:56, 4:66-5:8, 5:20-37.
after modifying the at least one parameter, call the API of the base OS with the at least one modified parameter;	Ex-1006 at 2, Figs. 3-7 (“The injected DLL’s function then examines the parameters that are passed to the API and modifies them <i>The injected DLL’s function then calls the real OS API with the modified parameters to perform the required function.”); see also id. at 3–7.</i>	Ex-1005, at Fig. 3-7, 3:34-4:56, 4:66-5:8, 5:20-37.

receive information from the API of the base OS;	Ex-1006 at 2–3, Figs. 3-7 (“ <i>Lastly, the injected DLL’s function returns to the calling function, returning any information from the OS API function.</i> This returned information is also modified to convert the file system or registry location information back from the virtual OS environment location.”); <i>see also id.</i> at 3–7.	Ex-1005, at Figs. 3-7, 3:34-4:56, 4:66-5:8, 5:20-37.
modify the information from the API of the base OS to convert the information back from a virtual OS environment location;	Ex-1006 at 2–3, Figs. 3-7 (“Lastly, the injected DLL’s function returns to the calling function, returning any information from the OS API function. <i>This returned information is also modified to convert the file system or registry location information back from the virtual OS environment location.</i> ”); <i>see also id.</i> at 3–7.	Ex-1005, at Figs. 3-7, 3:34-4:56, 4:66-5:8, 5:20-37.
return the information to the calling function of the application.	Ex-1006 at 2–3, Figs. 3-7 (“ <i>Lastly, the injected DLL’s function returns to the calling function, returning any information from the OS API function.</i> This returned information is also modified to convert the file system or registry location information back from the virtual OS environment location. By using this method, the injected DLL functions can execute code both before and after the OS API function.”); <i>see also id.</i> at 3–7.	Ex-1005, at Figs. 3-7, 3:34-4:56, 4:66-5:8, 5:20-37.

IV. LEGAL PRINCIPLES

24. IBM’s counsel has informed me of certain legal standards. I have applied these understandings in my analysis as detailed below. I am not an attorney and offer no legal opinions.

25. I have been informed and understand that a patent claim is invalid if it

is obvious, and that obviousness requires that the claim be obvious from the perspective of a person of ordinary skill in the relevant art at the time of the alleged invention. I have been informed and understand that a claim may be obvious in view of a single prior art reference, or may be obvious in view of a combination of two or more prior art references.

26. I have been informed and understand that the level of ordinary skill in the art may be determined by considering such factors as the education level of those working in the field, the sophistication of the technology, the types of problems encountered in the art, the prior art solutions to those problems, and the speed at which innovations are made. I have been informed and understand that not all of these factors need be considered and that one or more of these factors may control.

27. I have been informed and understand that an obviousness analysis is based on several factors, including the scope and content of the prior art, what differences, if any, existed between the alleged invention and the prior art, and the level of ordinary skill in the pertinent art at the time of the alleged invention.

28. I have been informed and understand that to determine whether a claim is obvious, it is appropriate to consider, among other factors:

29. (1) whether the teachings of the prior art references disclose known concepts combined in familiar ways, and when combined, would yield predictable results; (2) whether a person of ordinary skill in the art could implement a predictable

variation, and would see the benefit of doing so; (3) whether the claimed elements represent one of a limited number of known design choices, and would have a reasonable expectation of success by those skilled in the art; (4) whether a person of ordinary skill would have recognized a reason to combine known elements in the manner described in the claim; (5) whether there is some teaching or suggestion in the prior art to make the modification or combination of elements claimed in the patent; and (6) whether the innovation applies a known technique that had been used to improve a similar device or method in a similar way.

30. I further have been informed and understand that while it may be helpful to identify a reason for this combination, common sense should guide and no rigid requirement of finding a teaching, suggestion, or motivation to combine is required.

31. I further have been informed and understand that certain factors called secondary considerations may be considered in evaluating the obviousness of a claim. I understand that such secondary considerations include, among other things, commercial success of the alleged invention, skepticism of those having ordinary skill in the art at the time of the alleged invention, unexpected results of the alleged invention, any long-felt but unsolved need in the art that was satisfied by the alleged invention, the failure of others to make the alleged invention, praise of the alleged invention by those having ordinary skill in the art, and copying of the alleged

invention by others in the field. I have been informed and understand that there must be a nexus—a connection—between any such secondary considerations and the alleged invention. I also have been informed and understand that contemporaneous and independent invention by others is a secondary consideration tending to show obviousness.

32. I have been informed and understand that a prior art reference may expressly or inherently describe a limitation of a claim. I have been informed and understand that if the prior art necessarily includes or functions in accordance with a claim's limitation, then the prior art inherently discloses that limitation. I have been informed and understand that to establish inherency, the evidence must make clear that the missing descriptive matter is necessarily present in the item of prior art and that it would be so recognized by persons of ordinary skill in the art.

V. PERSON OF ORDINARY SKILL IN THE ART

33. I understand that there are multiple factors relevant to determining the level of ordinary skill in the pertinent art, including the educational level of active workers in the field at the time of the alleged invention, the sophistication of the technology, the type of problems encountered in the art, and the prior art solutions to those problem.

34. In determining the characteristics of a hypothetical person of ordinary skill in the art ("POSITA") of the '814 Patent at the time of the alleged invention, I

considered several things, including the type of problems encountered in this field and the rapidity with which innovations were made. I also considered the sophistication of the technology involved, the educational background and experience of those actively working in the field, and the level of education that would be necessary to understand the '814 Patent. I also, based on my own experience, placed myself back in the relevant periods of time and considered the state of the art and level of skill of computer scientists working in this field in those times

35. In my opinion, a POSITA in the subject matter of the '814 Patent would have had a bachelor's degree in computer science (or a similar degree) and at least two years of experience in operating systems or computer systems software. Additional education could offset less experience and vice versa, and a POSITA could have also obtained similar knowledge and experience through other means.

36. I have been a POSITA since at least 1991. The opinions I am offering are from the perspective of a POSITA at the time of the alleged invention, which I understand to be the 2002-2003 timeframe. For the purposes of my opinions below, I have been asked to assume that the relevant priority date for the Challenged Claims is September 15, 2003, and have applied the understanding of a POSITA as of that date.

VI. BACKGROUND OF THE '814 PATENT

A. Overview of the '814 Patent

37. The '814 Patent is titled “System for Containerization of Application Sets.”

38. The '814 Patent relates to computer software, or more specifically, the management and deployment of server applications.” Ex-1001, 1:14–16.

39. For example, the '814 Patent explains that computer system design results in various applications relying, or sharing, certain common resources, and that it is essential to safely and effectively provide those resources to applications. Ex-1001, 1:20–26.

40. While the '814 Patent notes that operating systems traditionally provide a mechanism for safely and effectively providing shared resources to applications, it also recognizes that one of limitations with this approach is that on any given computer system, “a collection of applications, each designed for a distinct function, must be separated with each application installed on an individual computer system.” *Id.*, 1:27–30.

41. The '814 Patent explains that this requirement that applications be separated is necessitated “by conflicts over shared resources, such as network port numbers that would otherwise occur,” “the requirement to securely separate data such as files contained on disk-based storage and/or applications between disparate users,” or “the reality that certain applications require a specific version of operating

system facilities and as such will not co-exist with applications that require another version.” *Id.* at 1:31–40.

42. As its alleged solution, the ’814 Patent describes the use of separate software “containers,” where a container is a group of files that includes at least one application and system files needed for the container to interact with the underlying OS kernel.

43. Thus, rather than to deploy each application on an separate individual computer platform, the ’814 Patent explains that applications can be run in containers, where the containers then run on the same computer platform. This made possible because applications running within a container have their own set of resources to minimize conflicts, secure files and data, and thus, allow multiple applications to co-exist on the same computer platform. Ex. 1005, 1:65-2:3.

B. Prosecution History of the ’814 Patent

44. I have reviewed the file history of the ’814 Patent (Ex-1002, FH). I have also reviewed the summary of the prosecution history in the Petition, and I agree with it.

C. Priority Date of the Challenged Claims

45. I understand that the priority date of a patent is used to determine whether a reference qualifies as prior art for the purposes of anticipation and obviousness. For the purposes of my analysis, I have been instructed to apply a

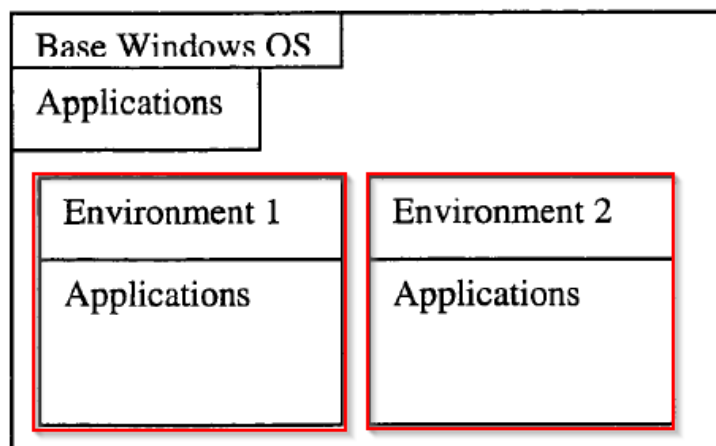
priority date of September 15, 2003 for the Challenged Claims. I am not offering an opinion as to the priority date the Challenged Claims are entitled to.

VII. BACKGROUND OF THE PRIOR ART

A. McMillan

46. McMillan is a U.S. Patent that issued on March 2, 2010, and is titled “Virtual OS Computing Environment.” Ex-1005. McMillan claims priority to U.S. Provisional Application No. 60/427,339 filed on November 18, 2002, titled the same. Like the ’814 Patent, McMillan discloses, among its many embodiments, the use of secure containers stored in a central repository (*i.e.*, central server) accessible by multiple different servers, where the servers can have the same or different base operating systems.

47. For example, in one embodiment, McMillan discloses multiple separate virtual operating system (OS) environments that work with base OS. *Id.*, 1:12–14. Each of these virtual OS environments is a “container” as described and claimed in the ’814 patent. Figure 1 of McMillan illustrates this by depicting a server with a base OS as well as two virtual environments (annotated in red):

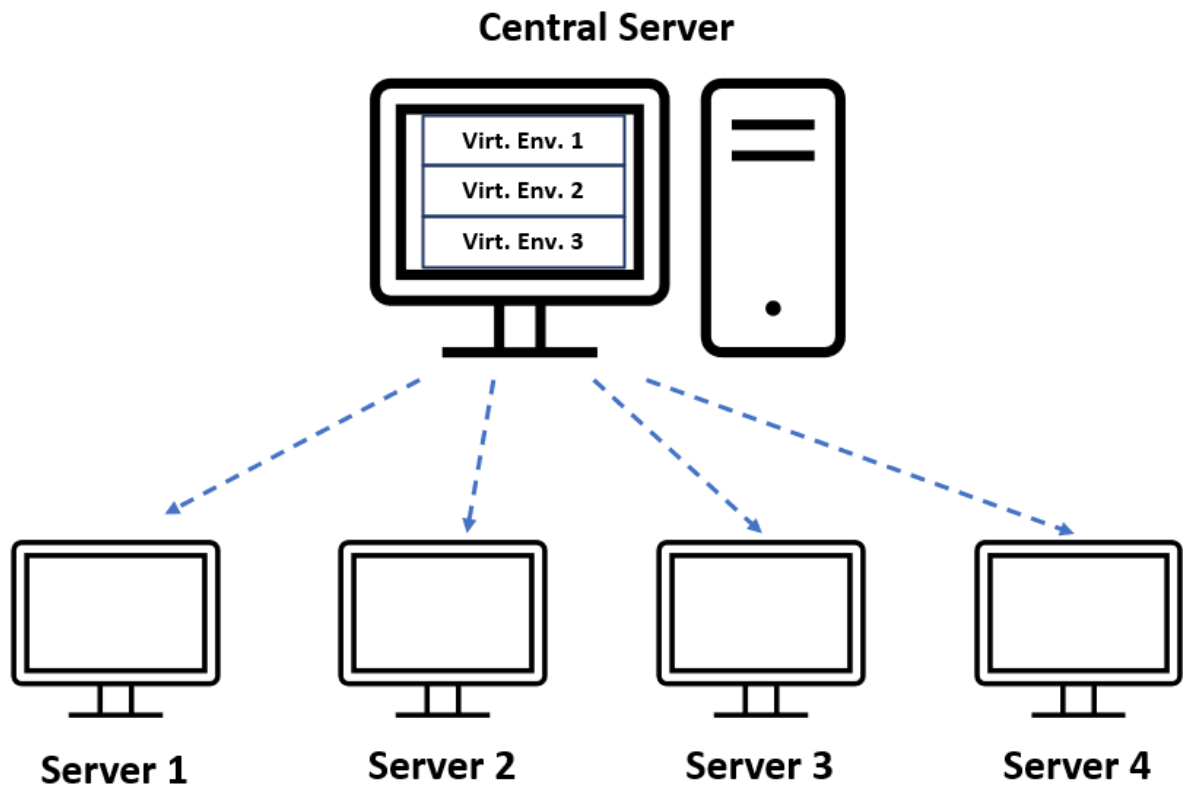


Ex-1005, Fig. 1 (annotated).

48. Each environment includes one or more applications as well as additional files required to run the applications, such as a copy of the base OS file system and registry. Ex-1005, 4:21-25. As a result, each environment runs independently of the others, such that changes to one environment do not affect the other environments. *Id.*, 2:55–64. McMillan’s environments are containers as defined by the ’814 patent, which explains that containers are merely “[a]n aggregate of files required to successfully execute a set of software applications on a computing platform.” Ex-1001, 2:23–25. The term “container” does not require or connote a specific type of data structure or format. These containers are also secure containers because each container is insulated from the other containers and the base OS. *See* Ex-1005, 2:15–19, 2:57-60; Ex-1001, 2:43–48.

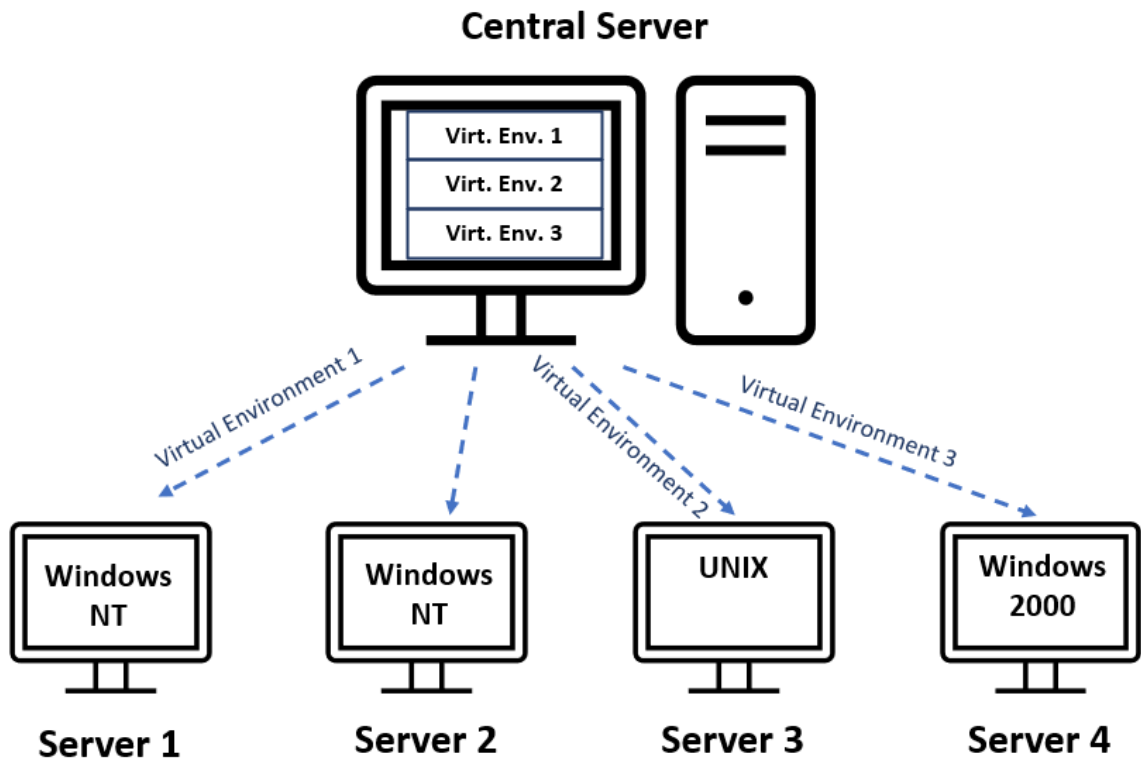
49. In addition to this single-server embodiment, McMillan also discloses that its environments (containers) may be stored in a central location accessible to

other multiple other computers, such as a central server. For example, McMillan explains that containers may be stored “in a separate or central location to be used by multiple computers or the same computer at different times.” Ex-1005, 4:57–65. To illustrate the foregoing, and because McMillan only describes its multi-server environment without providing a schematic, I have prepared the below schematic that illustrates McMillan’s teaching of an architecture where containers are centrally located on a central server and accessible by multiple other servers:



50. McMillan also recognizes that although it discloses embodiments using versions of the Windows OS, which at the time included versions such as Windows NT, Windows 98, and Windows 2000, embodiments are not limited to Windows,

and instead, may be used with other operating systems. *Id.*, 2:64–66 (“Although the invention is described in terms of a Windows® operating system environment, the approach is applicable to other operating systems through appropriate modification.”). Thus, each container may be configured towards a different operating system:



As depicted above, any given server that wants to execute a McMillan container compatible with the base OS of that server will communicate with the central server and retrieve the container for execution locally.

B. Schaefer

51. Schaefer is a U.S. Patent Pub. No. 2002/0174215 A1 filed on May 16,

2001 and published on November 21, 2002, and is titled “Operating System Abstraction and Protection Layer.” Ex-1008, Cover. Schaefer teaches the use of software containers, which Schaefer refers to as “private contexts.” As taught in Schaefer, the use of private contexts (containers) facilitates application execution without requiring changes to an underlying OS. *See, e.g.*, Ex-1008, Abstract (“The present invention provides a system for creating an application software environment without changing an operating system of a client computer.”), [0004] (same).

52. As Schaefer describes, a private context (container) includes an application to be executed, as well as an OS abstraction and protection layer, which Schaefer refers to as OS Guard. The OS Guard is interposed between one or more software applications and the operating system. Ex-1008, [0004], [0013], [0063]. Schaefer explains that the abstraction and protection layer is responsible for providing and securing virtual environments (*i.e.*, private contexts) by “present[ing] an environment to an application that appears to be an installation environment without performing an installation, whereby a ‘pseudo installation’ is created in which all of the settings are brought into a virtual environment at the time the application runs. Or in the case of an installed application, acts to dynamically modify the behavior of the application at run-time.” Ex-1008, [0007]; *see also id.*, [0014], [0027], [0040-41]; [0058-60].

53. Further, Schaefer explains that “any changes directly to the operating system are selectively made within the context of the running application and the abstraction and protection layer dynamically changes the virtual environment according to administrative settings.” Ex-1008, [0004]. This demonstrates that virtual environments in Schaefer are secure and separate from the base OS and other virtual environments, since the OS Guard taught by Schaefer prevents changes made to the base OS from being simultaneously and automatically applied to the virtual environment.

54. Schaefer teaches several purpose for its claimed virtual environments, including that “it prevents a running application from making changes to the client computer,” it “presents an environment to a running application that appears to be an installation environment without performing an installation,” and it “prevents information that may exist on the client/users machine from interfering with or modifying the behavior of an application.” Ex-1008, [0013]-[0015]. This further demonstrates that the virtual environments in Schaefer are secure and separate environments, such that the applications running within the virtual environments cannot interfere with the base OS and vice versa.

55. In my opinion, a private context in Schaefer is a “container” because it meets the ’814 Patent’s express definition for a “container,” which is “[a]n aggregate of files required to successfully execute a set of software applications on a computing

platform.” Ex-1001, 2:23–25. Specifically, each private context in Schaefer includes the application to be run as well as system files in the abstraction and protection layer. Ex-1008, [0007], [0014]; [0027], [0040-41]; [0058-60].

C. Gaines

56. Gaines is a U.S. Patent that issued on October 5, 1999, and is titled “Distributed and Portable Execution Environment.” Ex-1007. Gaines discloses a distributed execution environment comprising multiple servers running different OSs, where applications can be executed on any given server without concern for or knowledge of the underlying OS. Gaines explains that a “virtual operating system is capable of executing programs in a standardized programming language, which may be either an interpreted or compiled language, to provide the remote user with the ability to run programs that are host-independent.” Ex-1007, 4:13-18; *see also id.* 6:22-35. Gaines further recognizes that one of the advantages of its claimed invention, as compared to prior art where “the program to be executed on the virtual machine was still required to be tailored for that particular hardware and the services provided by the particular operating system software[,]” includes the provisioning of “a virtual execution environment for which programs could be written that are capable of execution on different computer hardware and with different operating systems.” Ex-1007, 3:62-4:2.

57. For example, Gaines’ Figure 3A depicts the claimed distributed

execution environment, illustrating that each host computer 101 (*i.e.*, each host server) is coupled to a network 131 and controlled by an operating system 103, in which “each set of computer hardware 102 may differ from each other, and each operating system 103 may also differ from each other.” Ex-1007, 12:60–62, Fig. 3A.

58. For a host computer on which applications are to be executed, Gaines teaches “provid[ing] a virtual execution environment for which programs could be written that are capable of execution on different computer hardware and with different operating systems.” Ex-1007, 3:66–4:2. Gaines facilitates the execution of an application across computers having different OSs through the use of a “server program [which] includes an interpreter for the (interpreted) programming language, a process control subsystem, and a virtual file subsystem.” Ex-1007, Abstract, Figs. 1 and 3A, 4:23-28.

59. Specifically, Gaines explains that “[t]he virtual operating system 141 includes a virtual application interface 144, having its own set of entry points 145, for the virtual applications 143 to call upon services from the virtual operating system [rather than the base OS].” Further, “[t]he virtual application interface 144 comprises a set of resource filters 146.” Ex-1007, 6:28-35. Gaines further explains that the resource filters perform several roles in provisioning a virtual environment for an application to successfully run within, including “translat[ing] each request

(by means of the virtual application interface 144) for virtual resources 142 into one or more requests (by means of the application interface 111) for actual resources 122[,]” “implement[ing] a virtual process model 152 comprising a set of virtual processes 153 organized in the virtual process model 152 name space,” “implement[ing] a virtual file system 162 comprising a set of virtual files (or virtual file system objects) 163 organized in a virtual file system 162 name space.” *Id.* at 6:41-9:35.

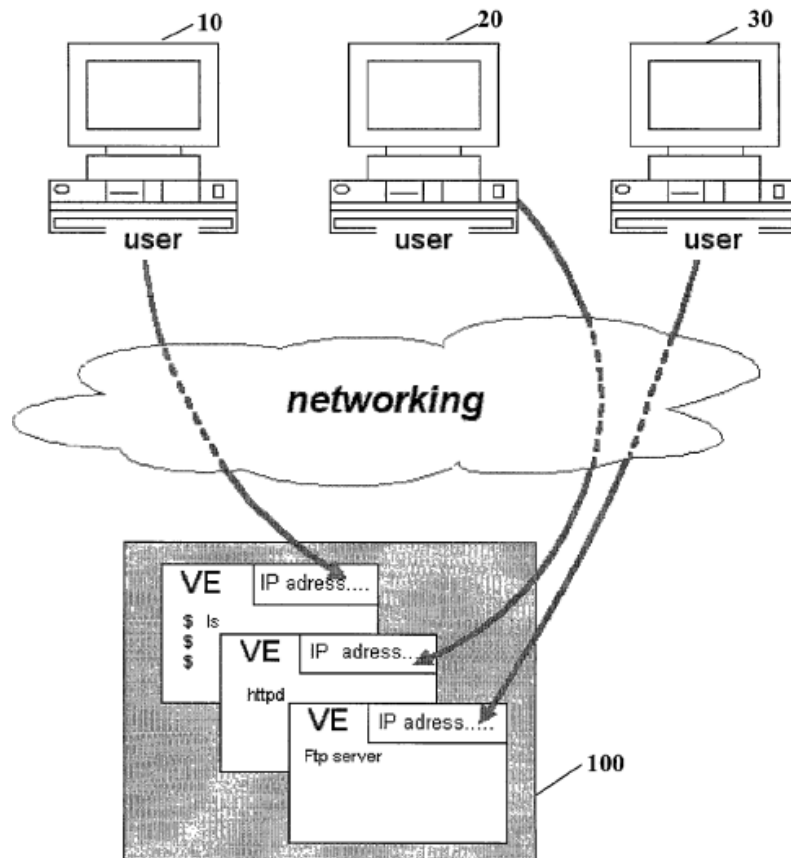
60. In addition to the above, “[t]he virtual operating system 141 [] comprises a language engine 191 that receives statements in a selected programming language and translates those statements into executable actions on the host computer 101.” Ex-1003, 9:36-45. “[T]he language engine 191 comprises an interpreter for the programming language” or in the alternative, “a compiler, or the interpreter may provide means for compiling all or part of a program in the selected programming language.” *Id.*; *see also* 9:59-10-7 (discussing interplay between the “language engine” and the “virtual application interface”).

61. Further, Gaines shows in its Figure 3A several host computers (*i.e.*, servers) on a network, each comprising a “transferable application.” Gaines explains that “[a] program (such as a transferable program 302 described herein) may be invoked by a user 201 either at a host computer 101 having a virtual operating system 141, at a host computer 101 without a virtual operating system 141, or at a host

computer 101 which has a virtual operating system 141 but which refuses to execute transferable programs 302 for users 201.” Ex-1007, 13:3-9. Thus, Gaines teaches a network through which that files can be transmitted across multiple computers.

D. Tormasov

62. Tormasov is titled “Virtual Computing Environment.” Ex-1009, Cover. Tormasov discloses a networked environment with multiple computers connected to a central server on a network:



Id. at Fig. 1; *see also id.*, [0024]. The hardware node (*i.e.*, a central server) stores a plurality of virtual environments (*i.e.*, containers) that are identified to each user

computer (*i.e.*, server) using unique IP addresses, as illustrated in Figure 1. *Id.*, [0024]. The unique IP address attributable to each virtual environment (*i.e.*, container) allows any given computer connected to the network to locate and retrieve a specific virtual environment on the hardware node. A POSITA would understand that the hardware node constitutes a central server because it is well-known that a network infrastructure includes nodes, which serve as connection points among devices (*i.e.*, computers / servers) connected to the network. Thus, in Figure 1 illustrated above, each of the user computers constitutes a node on the network, and each user computer (*i.e.*, node) is then able to establish a connection with the hardware node comprising the virtual environments (*i.e.*, central server).

VIII. CLAIM CONSTRUCTION

63. I understand that Petitioner and PO have exchanged claim constructions for certain terms of the '814 Patent but have not yet submitted claim construction briefings for any terms of the '814 Patent. I am unaware of any claim construction determinations related to the '814 Patent.

64. I further understand that Patent Owner has agreed to certain constructions in its litigation against Petitioner. I also understand that Patent Owner has proposed for certain claim terms a construction other than plain and ordinary meaning. I also consider obviousness under these constructions in my analysis below.

65. For the purposes of this proceeding, I have considered whether the Challenged Claims are invalid under their plain and ordinary meaning, the constructions agreed upon or argued in the Related IBM Litigation (VirtaMove, Corp. v. Hewlett Packard Enterprise Co., 2:24-cv-00093-JRG (E.D. Tex.) (lead case) and VirtaMove, Corp. v. International Business Machines Corp., 2:24-cv-00064-JRG (E.D. Tex.) (member case)), and the lexicography provided in the specification of the '058 Patent for relevant terms. In my opinion, the Challenged Claims are obvious under any of these constructions. The agreed upon terms and the constructions argued by Patent Owner for disputed constructions are shown below.

Term	Claim(s)	Agreed Construction
Preambles	1	The preambles are limiting.
processor	1	physical computer processor
secure container / secure container of application software	1, 2, 6, 9, 10, 31	“A container environment where each application set appears to have individual control of some critical system resources and/or where data within each application set is insulated from effects of other applications sets,” wherein “each container for use on a server is mutually exclusive of the other containers, such that read/write files within a container cannot be shared with other containers”
within a container	1	associated with a container
a kernel a set of associated local system files	1	a kernel and a set of associated local system files
a processor / the processor	1	The term and its antecedent refer to the same entity.

Term	Claim(s)	Agreed Construction
a plurality of servers / the servers	1	The term and its antecedent refer to the same entity.
one of the servers / the server	1	The term and its antecedent refer to the same entity.
operating systems / the operating systems	1	The term and its antecedent refer to the same entity.
a plurality of different operating systems / the plurality of different operating systems	1	The term and its antecedent refer to the same entity.
one or more of the executable applications / the one or more of the executable applications	1	The term and its antecedent refer to the same entity.
a plurality of secure containers of application software / the containers of application software / the plurality of secure containers of application software / the containers	1	The term and its antecedent refer to the same entity.

Ex-1014.

Term	Claim(s)	Patent Owner's Construction
disparate computing environments	1	"environments run by standalone computers"
system files	1, 10	No construction necessary; plain and ordinary meaning.

Id.

IX. INVALIDITY OPINIONS: '814 PATENT

66. In my opinion, the Challenged Claims of the '814 Patent are invalid based on the following references and combinations.

A. Ground 1: McMillan Alone or In View of Gaines and Tormasov

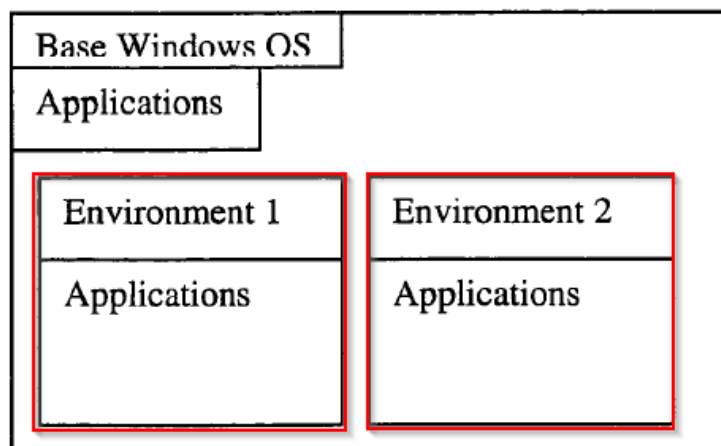
67. In my opinion, McMillan alone or in combination with Gaines and/or Tormasov renders claims 1, 2, 6, 9, and 10 obvious.

1. Claim 1

- a. 1 [Pre.a]: “In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel [and] a set of associated local system files compatible with the processor,”**

68. As I explain above in the Background of the Prior Art, McMillan is directed towards the use of secure containers stored in a central repository accessible by multiple different servers, where the servers can have the same or different base operating systems. In my opinion, McMillan alone or in view of Gaines renders obvious 1[Pre.a].

69. **First**, McMillan discloses or renders obvious a plurality of servers. For example, McMillan provides Figure 1, which depicts an embodiment comprising a single server having a base OS and two virtual OS environments:



Ex-1005, Fig. 1 (annotated).

70. In my opinion, these virtual OS environments are the claimed “containers” in limitation [1a], as discussed below.

71. Further, McMillan teaches that virtual OS environments may be stored in a central location, such as a central server, accessible to multiple other computers. The multiple other computers can then access and run any given virtual OS environment stored on that central server. Ex-1005, 4:60–65 (“It is therefore convenient to create a virtual OS environment that contains a set of applications configured in a specific way (or a ‘clean’ virtual OS environment that contains no applications yet) and store it in a separate or central location to be used by multiple computers . . .”).

72. In my opinion, each of these computers constitutes a server because it provides a service to the users of a computer. Ex-1001, 2:13-15 (“This invention provides a solution whereby a plurality of services can conveniently be installed on

one or more servers.”); *see also* Ex-1005, 4:60-65. That is, an application downloaded from the central server as part of a container can be installed and executed on the computer thereafter on behalf of the users. Ex-1005, 4:60-65.

73. **Second**, in my opinion, McMillan discloses or renders obvious that these multiple computers (servers) may have operating systems that differ. McMillan is directed at the use of virtual OS environments (containers) in order to enhance system usability and versatility. *See* Ex-1005, 2:64–66, 4:30–32, 4:60–65.

74. While McMillan discloses specific embodiments using a Windows OS, which at the time of McMillan was available in multiple different versions including Windows NT, Windows 98, Windows 2000, and Windows XP.

75. McMillan expressly teaches that its “approach is applicable to other operating systems through appropriate modification.” Ex-1005, 2:64–67 (“Although this invention is described in terms of a Windows® operating system environment, the approach is applicable to other operating systems through appropriate modification.”). McMillan also recognizes a number of major standard operating systems, including “Windows,” “Macintosh,” and “Unix,” and thus, a POSITA would understand McMillan’s teachings can be applied to at least any of these three different operating systems, given McMillan’s statement that its invention can be extend to other operating systems. Ex-1005, 1:44-56.

76. Further, a POSITA would understand that using McMillan’s virtual OS

environments (*i.e.*, containers) with multiple different operating systems would have the clear and obvious benefit of increasing the usability, versatility, and flexibility of McMillan's system by allowing virtual OS environments, as taught by McMillan, to be used with multiple different computers (and an increased number of users) regardless of their underlying OS distribution. At the time, computer systems could implement different base OS distributions, including those mentioned above—*i.e.*, Windows, MacIntosh, and Unix. A network infrastructure often included peer-to-peer connections among computers that use different operating systems—*i.e.*, one computer may run on a Unix OS distribution; another computer may run on a Windows OS distribution. Thus, a POSITA would have found it obvious to implement McMillan's teachings in a network comprising multiple computers, where each computer uses a different operating system, because these kinds of network architectures was common and well-known.

77. This common use of computers with multiple OSs in the same system is exemplified by Gaines, which discloses a distributed execution environment comprising multiple host computers, and that each host computer may execute different operating systems. For example, Gaines explains that “the host computer may comprise an IBM-compatible PC, a DEC Alpha, a Sun Workstation, an Apple Macintosh, or some other computer system” and that “[t]he operating system that executes on that host computer may comprise a variant of UNIX, DOS (from

Microsoft, IBM, and others), VMS (from Digital Equipment Corp.), MVS (from IBM), or System 7 (from Apple Computer).” Ex. 1007, 5:7-14.

78. In my opinion, a POSITA would have been motivated to apply Gaines’ teaching of the use of different operating systems to McMillan’s system in order to improve system efficiency, flexibility, and versatility. Similar to McMillan, Gaines teaches a system wherein secure virtual OS environments, within which software applications can execute and run, can be provided to host computers coupled to a network, where these computers may each have a different operating system. *See* Ex-1007, 5:24–28, Fig. 1 (showing a virtual OS environment within a host computer that is connected to a network), Fig. 3A (illustrating distributed execution environment). As already discussed above, Gaines’ teaching of multiple host computers, each with a different operating system, enhances the overall usability, flexibility, and versatility of Gaines’ system because it increases the number of users who can use Gaines’ virtual execution environments and the number of different applications that can be used.

79. In my opinion, a POSITA seeking to implement McMillan’s system would be naturally motivated to incorporate Gaines’ multi-operating-system approach in order to achieve these same universal goals of usability, flexibility, and versatility for any well-designed computer system. This would be accomplished by having multiple different virtual OS environments (OS) geared towards different

OSs stored in McMillan's system such that applications running within the virtual OS environments can be deployed on any given host computer, without regard to the underlying base OS distribution. Modifying McMillan to allow it to function with multiple different OSs would also be common sense and amount to the use of a known technique (*i.e.*, the use of multiple computers, each have different underlying base OS distributions) to improve similar systems (*i.e.*, Gaines' and McMillan's system) in the same way (*i.e.*, increasing usability, versatility, and flexibility of each respective system).

80. Further, in my opinion, combining the teachings of Gaines and McMillan would be a combination of prior art elements according to known methods to yield predictable results. Specifically, in my opinion, the use of multiple operating systems in a computer system was well known in the art and readily implementable by a POSITA. In fact, I routinely used multiple operating systems in a computer system in my research and in courses that I taught at Carnegie Mellon University (*e.g.*, providing a consistent development platform across different machines for the compiler optimization course, performing research on new bug-finding tools for device drivers, enabling new OS development on a live machine that is running a different OS, etc.). It would also be obvious to try this combination as one of a finite number of possible configurations—a system can either have a common operating system on all servers, or there can be differences among them. Modifying

McMillan's system to use multiple operating systems would involve routine and well-known computer programming techniques with no requirements for hardware modification because it would only require utilizing different commercially available OSs on different computers that were commonly deployed in practice at the time. It would similarly be routine to have these computers communicate over a network, particularly where peer nodes (*i.e.*, computers/servers) on a distributed network use a standard format and a common interface for communicating with another peer node. As such, a POSITA would have a reasonable expectation of success in implementing such a modification with predictable results.

81. **Third**, McMillan discloses or renders obvious that its multiple computers (servers) operate in disparate computing environments. My opinion is the same under both parties' constructions of "disparate computing environments." Specifically, McMillan meets PO's construction of "environments run by standalone computers," because McMillan discloses "multiple computers" that are separate from each other, and each computer runs virtual OS environments independently of the other computers. Ex-1005, 4:57–65. McMillan also meets Petitioner's proposed construction of "[e]nvironments where computers are stand-alone or where there are plural computers and where they are unrelated," which is based on lexicography (Ex-1001, 2:17–19), not only for this same reason, but also because McMillan discloses the use of plural computers having no relationship with each other (*i.e.*, they are

standalone computers). Ex-1005, 4:60–65.

82. **Fourth**, McMillan discloses or renders obvious that each of its multiple computers (servers) includes a processor and an operating system including a kernel [and] a set of associated local system files compatible with the processor.

83. In my opinion, a POSITA would understand and find obvious that a computer includes a physical computer processor, because the use of a processor is the defining feature that renders a computer a computer. Ex-1009 at 45 (“The ‘brain’ of the computer is the CPU.”), Figs. 1-5 (illustrating “CPU”, *i.e.*, a processor). Further, in my opinion, a POSITA would also understand that McMillan necessarily contemplates the presence of a physical processor, given its multiple references to “running” an application and similar language—*i.e.*, processors run software such as applications.

84. Further, McMillan expressly discloses that each of its computers must have a base OS in order to run a virtual OS environment. Specifically, because McMillan expressly teaches that applications can run either in a virtual OS environment, or alternatively, use the base OS, McMillan, discloses that each computer contains an OS (*i.e.*, the base OS). *See* Ex-1005, 2:55-57 (“This invention is directed to the creation of multiple, semi-independent virtual OS environments within a single operating system (OS).”), 3:4–5 (“all of the applications run on a single OS desktop.”); 3:17-33 (“This DLL first must determine whether the current

application should be run under one of the possible virtual OS environments or the base OS.”).

85. In my opinion, a POSITA would understand and find obvious that any OS has a kernel, because a kernel is the part of the OS that controls interactions between a computer’s hardware and the software executing on a computer. For example, this is explained in MODERN OPERATING SYSTEMS (2d ed. 2001) by Andrew S. Tanenbaum, a well-known textbook from 2001. Ex-1009 at 461, Fig. 11–7.

86. McMillan also discloses that a base OS has associated system files, which are present in all operating systems and were well-known to a POSITA. Ex-1009 at 62. For example, McMillan explicitly discloses that each computer (*i.e.*, server) includes a base OS file system and registry, which McMillan illustrates in Figure 2:

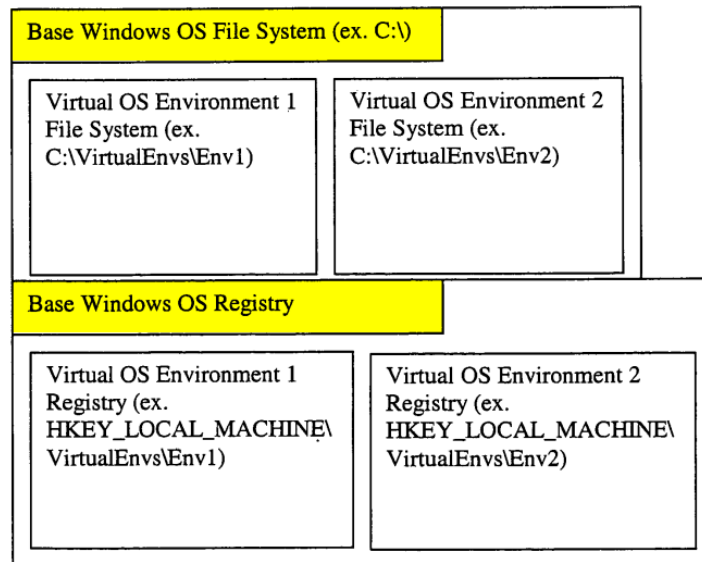


FIGURE 2

Fig. 2 (annotated).

87. As McMillan explains, an application requires access to a file system and registry in order to obtain files, services, and/or other information that are necessary for the application to run effectively. Ex-1005, 1:64-2:6, 3:17-30, 4:21-32; *see also id.*, 4:15-16, 4:21-25, 5:20-23.

88. In my opinion, a POSITA would understand and find it obvious that the base OS file system and registry include “system files provided within an operating system and which are available to applications as shared libraries and configuration files.” Ex-1001, 2:52-54; *see also id.*, 5:20-26. A POSITA would understand this because the base OS file system provides services, configuration files, and other resources to an application. *See, e.g.*, Ex-1009, 771-852 (discussing Windows 2000 file systems and system files); *see also, e.g., id.*, 5-6 (discussing the operating system

as a “resource manager”), 44-56 (discussing system calls, including Win32 API calls). Indeed, in McMillan, applications directed to the base OS would access system files from the base OS file system and registry. *See* Ex-1005, 3:11-48. Applications directed to virtual OS environments access those same system files because, although they access the virtual file system and registry, “a copy of the base OS file system and registry must be created in the virtual OS environment file system and registry before any applications may be executed in the virtual OS environment.” Ex-1005, 4:21-25.

89. Further, a POSITA would understand McMillan’s system files to include dynamically-linked libraries (“DLLs”), or shared libraries, which applications can link to thereafter, because DLLs are used to efficiently provide applications with services to access portions of any given file system. *See* Ex-1005, 1:64-2:6, 3:1-10, 3:17-30, 4:33-36. Indeed, a POSITA would understand that, in a Windows OS environment, as described by McMillan, applications make system calls to request services from the operating system, which perform specific functions, such as file access, memory management, network communication, and more. In a Windows OS environment, services can be provided through DLLs, which, at least in the Windows context, are files residing in system directories like “System32” within the base Windows OS file system. *See, e.g.*, Ex-1009, 771-852 (discussing Windows 2000 file systems and system files, including DLLs); *see also*,

e.g., id., 5-6 (discussing the operating system as a “resource manager”), 44-56 (discussing system calls, including Win32 API calls). Thus, a POSITA would understand that in McMillan’s system, applications can make system calls to OS APIs, such as Win32 API for example, to access system files, such executable files or DLLs, which would reside in a directory within the base Windows OS file system.

90. A POSITA would also understand that the base OS file system and registry could provide system files (*i.e.*, services) to an application, or alternatively, those system files (*i.e.*, services) could instead be wrapped and be provided to an application through the use of a dynamically-linked library (“DLL”), or shared library. *See* Ex-1005, 1:64-2:6, 3:17-48, 4:33-36. Further, a POSITA would recognize that McMillan’s express disclosure of the use of system calls and APIs, which are implicated in the process of accessing the file system and registry, to teach the presence of system files within the base OS file system and registry, as system calls and APIs are integral to the process by which an application would access system files, including services, in the file system. *See* Ex-1005, 3:17-47, 4:21-5:8.

b. 1 [Pre.b]: “a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment”

91. In my opinion, McMillan discloses or renders obvious 1[Pre.b].

92. **First**, McMillan discloses or renders obvious a method of providing at least some of the servers in the system with secure, executable applications related

to a service.

93. As discussed previously, McMillan teaches the ability to “save, store, and load entire virtual OS environments,” such as “in a separate or central location to be used by multiple computers.” Ex-1005, 4:57–65 (“It is therefore convenient to create a virtual OS environment that contains a set of applications configured in a specific way (or a ‘clean’ virtual OS environment that contains no applications yet) and store it in a separate or central location to be used by multiple computers . . .”). Further, as McMillan explains, these virtual OS environments (*i.e.*, containers) include secure, executable applications related to a service, such as NotePad and WordPad, which relate to a service such as word processing. *See id.*, 4:25–30 (“any built-in OS applications that are run by applications within the virtual OS environment (such as Notepad™ text editor, Wordpad™ word processor, or Internet Explorer® internet browser, all from Microsoft Corporation) will be executed from the virtual OS environment copy of that application.”). Further, these applications are secure because the applications execute within McMillan’s virtual OS environments “run independently of each other” and the base OS. *See* Ex-1005, 2:15–19.

94. **Second**, McMillan teaches that these applications are executed in a secure environment. McMillan’s virtual OS environments (*i.e.*, containers) are secure environments by virtue of their independent and isolated qualities.

Specifically, these virtual OS environments are independent and isolated environments (*i.e.*, secure environments), as indicated by their inability to affect the base OS or any other environment (*i.e.*, secure environments). Ex-1005, 2:57-60. McMillan also expressly indicates that its virtual OS environments are secure environments by explaining that “[t]he file system and registry information for each environment is independent of the base OS and other environments” and that the environments’ applications “will run independently of each other.” Ex-1005, 2:60-64; *see also id.*, Cl. 4. In my opinion, this is consistent with the teachings of the ’814 Patent, where applications execute in a secure environment as a result of “associating applications with secure containers [*i.e.*, virtual OS environments]” such that “[a]pplications are segregated and execute with their own copies of files and with a unique identity.” Ex-1001, 7:4–10.

95. McMillan also teaches that “[w]hen applications attempt to access the file system or registry, the attempt is redirected to the virtual file system or registry instead of the base OS location.” Ex-1005, 3:11-16. This indicates that applications in the McMillan’s virtual OS environments have their own copies of files, which are provided by the virtual OS file system and registry of the specific virtual OS environment in which they are running. *See also* Ex-1005, 21-25 (“[A] copy of the base OS file system and registry must be created in the virtual OS environment file system and registry before any applications may be executed in the virtual OS

environment.”).

- c. **1 [Pre.c]: “wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:”**

96. In my opinion, McMillan discloses or renders obvious 1[Pre.c].

97. **First**, McMillan discloses or renders obvious that “the applications each include an object executable ... for performing a task related to the service.” For example, McMillan teaches that applications running in its virtual OS environments include applications “such as Notepad™ text editor, Wordpad™ word processor, or Internet Explorer® internet browser, all from Microsoft Corporation” which “will be executed from the virtual OS environment copy of that application.” Ex-1005, 4:25–30, *see also id.*, 4:30–32. These applications include an EXE, or an executable file, which a POSITA would understand to be an object executable. Ex-1005, 3:59-61, 2:3-5. A POSITA would understand that McMillan’s EXE file is an object executable, because it is written in binary format to be executed by the processor of a computer. *See* Ex-1009 at 814; *see also id.*, 267; Ex-1010 at 44. A POSITA would further understand that these applications (*i.e.*, at least Notepad™ text editor, Wordpad™ word processor, or Internet Explorer® internet browser), which include an object executable, are for performing a task related to the service. For example, a service provided by Notepad™ text editor or Wordpad™ word processor can include editing a document, and specific tasks related to this service

can include typing words, font formatting, and text manipulation, among many others.

98. **Second**, McMillan discloses or renders obvious that the applications run in the virtual OS environments can be executed “by at least some of the different operating systems.” In order to illustrate this, consider a system of five servers, including two servers running Unix, two servers running Windows NT, and another running Windows 2000. By the time McMillan’s provisional application was filed in November 18, 2002, these operating systems were commercially available and well-known. Within the foregoing example system, which would be typical of common deployments of the time, and where there are servers having different underlying base OSs, an application written for Windows NT can be executed by one server running Windows NT, or by at least two of the servers, that is, both servers running Windows NT, which are “at least some of the different operating systems.” As previously discussed, McMillan discloses that its virtual OS environments (*i.e.*, containers) can be saved, stored, and loaded, or rather, more specifically, that they can be stored in and downloaded from a central location, and then used by multiple computers. *See* Ex-1005, 4:57–65. Thus, a POSITA would understand that McMillan discloses the ability that two different computers, each having its own underlying base OS distribution, (*i.e.*, both running different instances of Windows NT) can download a virtual OS environment from a central

location (memory) to execute the applications as disclosed in McMillan. *See* Ex-1005, 4:21–32, Figs. 1-2.

99. Further, McMillan discusses in depth the use of the OS Application Programming Interface (API). A POSITA would understand the OS API referenced by McMillan to be the Windows Win32 API given that McMillan’s disclosed embodiments are all based on Windows. *See* Ex-1005, 3:17–48, 4:33–56, Figs. 3, 5, 6. It was well known to a POSITA that the Win32 API “allows programs to run on almost all versions Windows,” including Windows NT and Windows 2000:

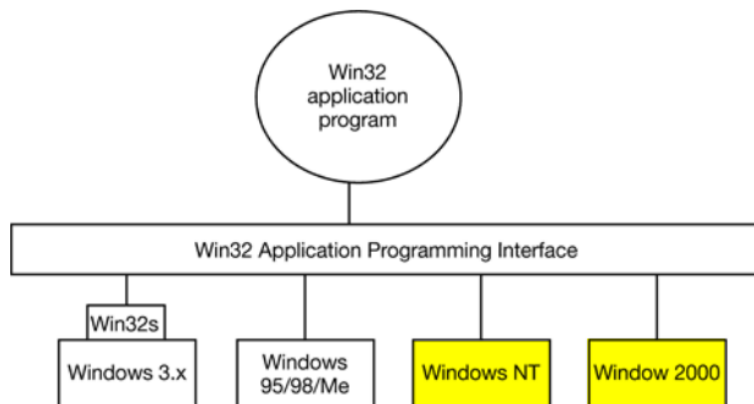


Figure 11-4. The Win32 API allows programs to run on almost all versions of Windows.

Ex-1009 at 455, Fig. 11-4 (annotated). Because Windows operating systems are backwards compatible, a POSITA would understand and find obvious that an application designed for Windows NT can be run on Windows NT and Windows 2000. Thus, a POSITA would understand that McMillan discloses the ability of multiple computers having different underlying base Windows OS distributions (*i.e.*, one computer running Windows NT and another running Windows 2000) to run the

same application.

- d. **1[a]: “storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers;”**

100. In my opinion, McMillan renders obvious 1[a].

101. **First**, McMillan discloses “storing in memory accessible to at least some of the servers a plurality of secure containers of application software.” Specifically, McMillan discloses storing its virtual OS environments (*i.e.*, containers) in a central location that is accessible to multiple computers (*i.e.*, servers), as discussed in Section IX.A.1.a above. In my opinion, a POSITA would understand that these virtual OS environments are stored in memory (*i.e.*, hard disk or other forms of storage) in the central location because these virtual OS environments are software. As I previously discussed, these virtual OS environments are software by virtue of them constituting a plurality of secure containers of application software (*i.e.*, secure, isolated environments within which applications can execute and run).

102. **Second**, McMillan discloses “each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications.” McMillan teaches that the virtual OS environments

(i.e., containers) contain a copy of a subset of the base OS file system and registry, further explaining that “any applications that are run from under the virtual OS environment file system (ex: C:\VirtualEnvs\Env1) are always redirected into that virtual OS environment.” Ex-1005, 4:4–16; *see also id.*, 4:21–30, Fig. 5. In other words, applications executing within McMillan’s virtual OS environments use “associated system files required to execute the one or more applications” because by accessing the virtual OS environment file system, the applications access copies of the base OS system files. Indeed, as previously explained, applications attempting to access the base OS file system, instead, are redirected to the virtual OS file system to access copies of the base OS system files instead.

103. **Third**, McMillan discloses or renders obvious that the secure application containers are “for use with a local kernel residing permanently on one of the servers.” For example, McMillan teaches that “[t]his invention is directed to the creation of multiple, semi-independent virtual OS environments within a single operating system (OS).” Ex-1005, 1:12-14; 2:10-13. McMillan also teaches that the “virtual OS environment is achieved by creating independent locations for file systems and registry within the base OS file system and registry.” Ex-1005, 3:11-13; Fig. 2. In my opinion, a POSITA would understand that the virtual OS environments, although having their own file systems and registries, must still use the base OS kernel to run, hence, McMillan’s disclosure of “semi-independent

virtual OS environments within a single operating system.”

104. Further, McMillan teaches that “[t]he applications running within the virtual OS environments still share the base OS attributes such as networking information, user login rights, services, hardware information, and the Windows clipboard.” Ex-1005, 3:1–4; *see also id.*, Cl. 2 In my opinion, a POSITA would understand and find obvious that networking information and hardware information are types of information within the kernel. Ex-1011 (Berger) at [0005]; Ex-1010 (Solomon) at 40. Further, in my opinion, a POSITA would understand that “applications running within virtual OS environments” could only “share the base OS attributes” where the virtual OS environments run on top of, and thus are used with, a base OS kernel. Additionally, McMillan consistently references the use of APIs, which in McMillan’s primarily Windows-based system, a POSITA would understand to be the Win32 API. A POSITA would understand that part of Win32 API is run in kernel mode and interacts with the kernel. *See* Ex-1010 at 41 (illustrating “Win32” in “kernel mode”).

e. 1[b]: “wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems,”

105. In my opinion, McMillan alone or in view of Gaines renders obvious 1[b].

106. As discussed previously, McMillan teaches that its virtual OS

environments (*i.e.*, secure containers) can be used for different operating systems by multiple computers. In my opinion, a POSITA would have understood and found obvious that the set of associated system files within McMillan's virtual OS environment, which are the copies of the base OS file system and registry, would be compatible with a local kernel of at least some of the plurality of different operating systems. To illustrate this, once again, consider a configuration of five servers, all of which are connected to a central server through a network, where two servers run Unix, two servers run Windows NT, and another server runs Windows 2000.

107. In a virtual OS environment (*i.e.*, container) designed for operation with a base Windows NT OS, the virtual OS environment would include system files compatible with the local kernel of at least the computers running Windows NT as a underlying base OS distribution. These two instances of Windows NT are "at least some of the plurality of different operating systems" within the configuration set forth above. Additionally, the set of associated system files would be compatible with the local kernel of the computers running Windows 2000 as a base OS, given that, as previously explained, both Windows NT and Windows 2000 would utilize Win32 API, which "allows program to run on almost all versions of Windows." *See* Ex-1005, 3:34–48 (virtual environments use injected DLL to modify parameters and call the base OS API); Ex-1009 at 455, Fig. 11-4 ("The Win32 API allows programs to run on almost all versions of Windows."), (illustrating Windows NT and Windows

2000); *see also* Ex-1005, Figs. 5, 6B.

108. Further, Gaines expressly teaches that a “virtual operating system is capable of executing programs in a standardized programming language, which may be either an interpreted or compiled language, to provide the remote user with the ability to run programs that are host-independent.” Ex-1007, 4:13-18; *see also id.*, 6:22-35. Gaines further teaches a plurality of computers with different operating systems, even identifying a broad range of operating systems that may execute on any given host computer, and that “[t]he host virtual operating system may be programmed for any of these hardware and operating system software.” Ex-1007, 5:1-16, 3:66-4:2 (recognizing the advantage of having “a virtual execution environment for which programs could be written that are capable of execution on different computer hardware and with different operating systems”).

109. In my opinion, it would have been understood and obvious to a POSITA that it could combine Gaines’ teaching of its “virtual application interface,” “resource filters,” and “language engine” with McMillan’s virtual OS environments, allowing the same container to be used on different operating systems such as Windows and Unix. *See* Ex-1008, Abstract, Figs. 1, 3A, 4:23-28 (“server program [which provides the virtual host operating system and] includes an interpreter for the (interpreted) programming language, a process control subsystem, and a virtual file subsystem.”), 6:28-9:35 (discussing the “virtual application interface” and “resource

filters”), 9:36-10-7 (discussing the “language engine” and the interplay between the “language engine” and the “virtual application interface”). As explained previously, the ability to run the same application on multiple different host computers, each having a different underlying base OS would confer several benefits, including the benefit of increasing the usability, versatility, and flexibility of McMillan’s system.

110. Combining Gaines’ teaching with McMillan’s teachings in the manner explained above would have been an application of a known technology to McMillan with predictable results (*i.e.*, enhanced usability, versatility, and flexibility of McMillan’s system). This combination also would have been obvious to try as one of a finite number of implementations. In my opinion, a POSITA would also have had a reasonable expectation of success, given that implementation would involve only software programming and no hardware modifications.

f. 1[c]: “the containers of application software excluding a kernel, wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server,”

111. In my opinion, McMillan renders obvious 1[c].

112. **First**, McMillan teaches that its virtual OS environments (*i.e.*, containers) exclude a kernel. Indeed, as explained already above, virtual OS environments as taught by McMillan use the base OS kernel because McMillan teaches that “[t]his invention is directed to the creation of multiple, semi-

independent virtual OS environments within a single operating system (OS).” Ex-1005, 2:10-13. McMillan also teaches that the “virtual OS environment is achieved by creating independent locations for file systems and registry within the base OS file system and registry.” Ex-1005, 3:11-13, Fig. 1. In my opinion, a POSITA would understand that the virtual OS environments, although having their own file systems and registries, must still use the base OS kernel to run, hence, McMillan’s disclosure of “semi-independent virtual OS environments within a single operating system.” That is, the virtual OS environments do not themselves have their own respective kernels on top of the base OS kernel.

113. McMillan further explains that “[t]he applications running within the virtual OS environments still share the base OS attributes such as networking information, user login rights, services, hardware information, and the Windows clipboard.” Ex-1005, 3:1–4; *see also id.*, Cl. 2. In my opinion, a POSITA would understand and find it obvious that at least networking information and hardware information are types of information provided by the kernel because networking information manages how the computer itself connects to the network and hardware information manages how the software on the computer interfaces with the hardware components of the computer. Ex-1011, [0005]; Ex-1010 at 40. Thus, if the virtual OS environments (*i.e.*, containers) possessed their own kernel, they would not need to rely on the base OS for this information. Additionally, a POSITA would

understand McMillan consistent reference to the use of APIs to be referring to the Win32 API, given that McMillan is primarily a Windows-based system. A POSITA would understand that part of Win32 API is run in kernel mode and located outside of the containers because system calls to Win32 API may require access to kernel-level resources, despite the fact that applications are making these calls within virtual OS environments. *See* Ex-1010 at 41 (illustrating “Win32” in “kernel mode”).

114. **Second**, McMillan’s virtual environments (*i.e.*, containers) stored in memory on the central server utilize copies of certain base OS system files and registries in running the application software in place of the associated local system files that remain resident on the server, because they use the container files and not the corresponding kernel files. Ex-1005, 4:21-25, Fig. 2 (illustrating Base OS File System and Registry), Fig. 7, Cl. 10; *see also supra* Section IX.A.1.e; Ex-1001, 10:18-34 (“Container files include [] those files provided by an operating system distribution . . . These files can be obtained by using a compute platform upon which the application of interest has been installed.”), 10:41-46 (“In the first method of container creation a ‘skeleton’ file system is used. This is a copy of the system files provided with a given operating system (OS) distribution . . . The skeleton file system includes the OS provided system files before an application is installed.”).

- g. 1[d]: “wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server,”**

115. In my opinion, McMillan renders obvious 1[d].

116. As I previously discussed for limitation 1[c], McMillan teaches that each of its virtual OS environment (*i.e.*, containers of application software) utilizes copies of certain base OS system files (*i.e.*, associated local system files that remain resident on the server) housed in the virtual OS file system rather than the base OS system files housed in the base OS file system.

- h. 1[e]: “and wherein the application software cannot be shared between the plurality of secure containers of application software, and wherein each of the containers has a unique root file system that is different from an operating system's root file system.”**

117. In my opinion, McMillan renders obvious 1[e].

118. **First**, McMillan discloses or renders obvious “wherein the application software cannot be shared between the plurality of secure containers of application software.” For example, McMillan teaches that each of its virtual OS environments is an independent and isolated environment, insulated from activities conducted within other OS environments in McMillan’s system, such that “[a] change made in one environment does not affect the main OS or any other environment.” Ex-1005, 2:57–58; *see also id.*, 2:59–60 (“[E]ach virtual OS environment appears to be an independent OS for the applications running within it.”), 2:62–64 (“Each of the

environments can contain a group of installed applications that will run independently of each other.”); Cls. 3, 4, 6. Indeed, in McMillan, an application is specific to a virtual environment, and not shared between environments, as depicted by Figure 1 for example, which shows each virtual environment having its own application sets. *See* Ex. 1005, Fig. 1, 2:62-64. Each of McMillan’s virtual OS environments also receives copies of certain base OS system files, further indicating the independent and isolated nature of each virtual OS environment. Ex-1005, 4:21-25. Thus, in my opinion, a POSITA would understand from McMillan’s teachings that the application software running within each virtual OS environment (*i.e.*, container) cannot be shared with other virtual OS environments (*i.e.*, containers), because the virtual OS environments run independently of each other, and as discussed below, have their own respective file system and registry information.

119. **Second**, McMillan discloses or renders obvious “wherein each of the containers has a unique root file system that is different from an operating system's root file system.” For example, as previously discussed, McMillan teaches that each virtual OS environment (*i.e.*, container) uses its own virtual OS file system, which include a separate copy of the base OS files system and registry (which further includes separate directory locations), thereby indicating that each virtual OS file system is a unique root file system. *See* Ex-1005, 4:21-25, Fig. 5. In other words, the virtual OS environments do not share the base OS file system, nor do they share

a virtual OS file system; they each have their own virtual OS file system distinct from other file systems – virtual or base. McMillan’s teachings support this, as McMillan explains that “[t]he file system and registry information for each environment is independent of the base OS and other environments.” Ex-1005, 2:60–62; *see also id.*, 3:11–13, 3:61–64, Cl. 1, Figs. 2, 4, 5, 7 (illustrating independent directories for file systems and registries for virtual OS environments).

2. Claim 2: “A method as defined in claim 1, wherein each container has an execution file associated therewith for starting the one or more applications.”

120. As I previously discussed, Claim 1 is obvious, and in my opinion claim 2 is obvious.

121. As previously discussed, McMillan discloses that the applications running in its virtual OS environments (*i.e.*, containers) include applications “such as Notepad™ text editor, Wordpad™ word processor, or Internet Explorer® internet browser, all from Microsoft Corporation” which “will be executed from the virtual OS environment copy of that application.” Ex-1005, 4:25–32. McMillan also explains that each application has an EXE, or an executable, that when run, starts the application. Ex-1005, 3:59-61 (referencing “application EXE”). A POSITA would understand and find obvious that “an application EXE” is an execution file associated with starting the given application. Ex-1009 at 814; *see also id.*, 267; Ex-1010 at 44 (“Each executable image (.EXE).”). The purpose of an application

executables (*i.e.*, execution files) is to provide the operating system with instructions for running an application program on the system. Without such instructions, the computer would not be able to run the given application. Further, executable files are necessary to coordinate interactions between the application program and the operating system, especially at run-time, where the operating system often provisions resources to the application for it to run effectively.

3. Claim 6: “A method as defined in claim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.”

122. As I previously discussed, claim 2 is obvious. In my opinion, claim 6 is also obvious.

123. **First**, McMillan teaches that each virtual OS environment (container) is assigned a unique identity. McMillan provides Figures 1 and 2, which demonstrate that each virtual OS environment (*i.e.*, container) includes a unique name, or identity (*i.e.*, Virtual OS Environment 1 (Env1) or Virtual OS Environment 2 (Env2)):

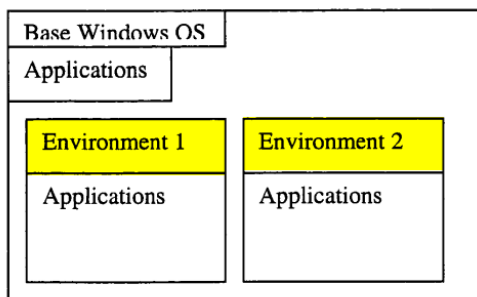


FIGURE 1

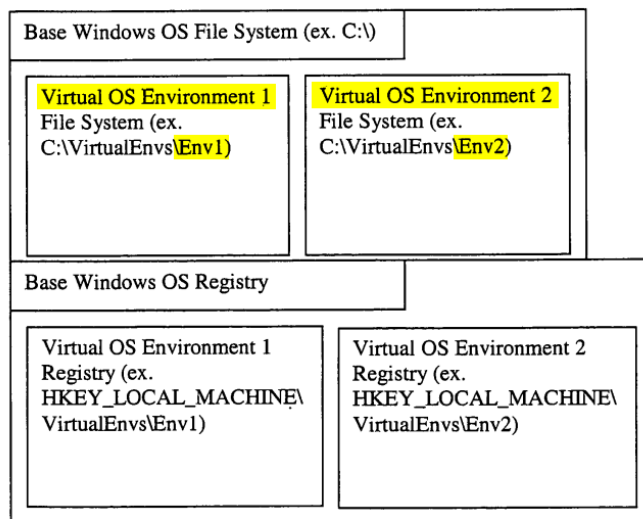


FIGURE 2

Ex-1005, Figs. 1, 2 (annotated); *see also id.*, 4:4–7 (“applications that are run from under the virtual OS environment file system (ex: C:\VirtualEnvs\Env1) are always redirected into that virtual OS environment.”); *see also* Ex-1005, Figs. 4–6. This is consistent with McMillan’s teaching that its virtual OS environments (*i.e.*, containers) are specific and independent of one another. Ex-1005, 2:57–58; *see also id.*, 2:59–60 (“[E]ach virtual OS environment appears to be an independent OS for the applications running within it.”), 2:62–64 (“Each of the environments can contain a group of installed applications that will run independently of each other.”); *see also id.*, 3:11–13, 3:57–64, Cl. 1, Figs. 2, 4 (illustrating independent directories for file systems and registries for virtual OS environments).

124. For example, McMillan explains that “[t]he base OS contains a list of directories where applications should be redirected to specific virtual OS

environments[,]” indicating that the virtual OS environments are distinctly identifiable from one another. Ex-1005, 3:57–61. McMillan further explains that “[a]ny application EXE that is executed in this directory or optionally a child of this directory will run under a specific virtual OS environment[,]” again indicating that the virtual OS environments are individually distinct and specific, and thus, have their own unique identity. *Id.* Gaines also teaches assigning a unique identity to particular copies of the virtual OS. *See* Ex-1007, 11:17-22.

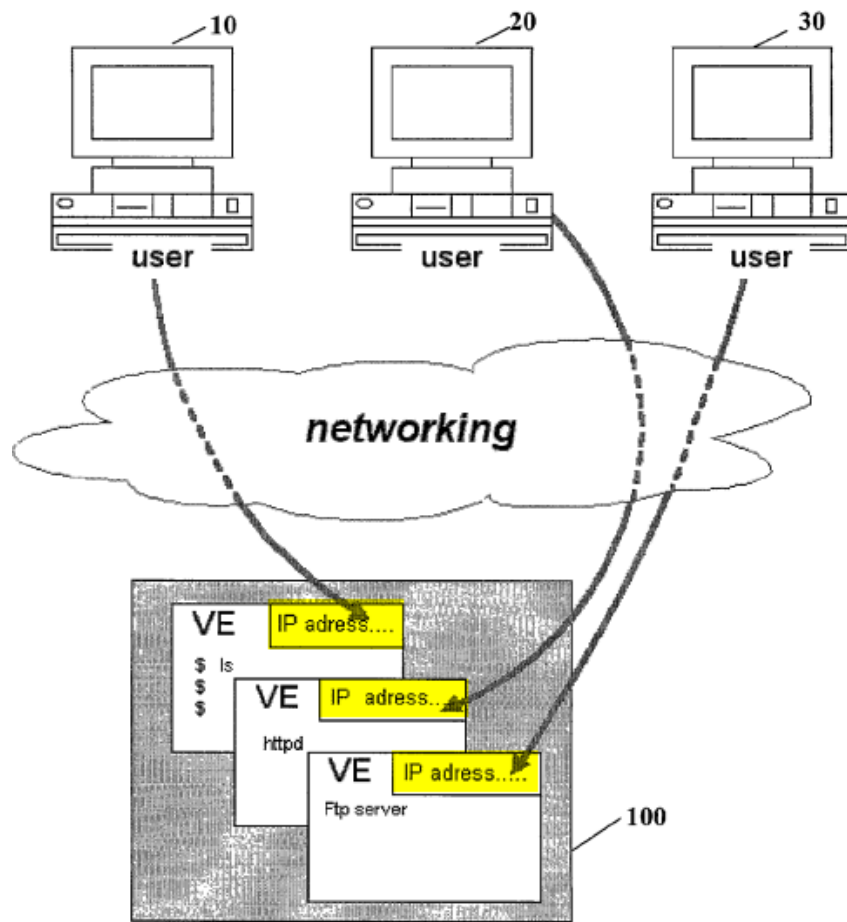
125. **Second**, McMillan discloses or renders obvious that each unique identity further includes at least a host name, MAC address, and/or IP address. McMillan explains that its environments are identified in a specific virtual directory, such as “C:\VirtualEnvs” or on a specific machine, such as “HKEY_LOCAL_MACHINE\VirtualEnvs.” These are host names because they are names that identify where the containers are hosted. Further, a POSITA would understand that on Windows OSs, a Windows parameter known as “hostname” is found in the Windows registry and is used to identify the computer (*i.e.*, server) hosting any given container. *See, e.g.*, Ex-1009, 774-777 (“The first key (*i.e.*, directory [included in the Windows 2000 Registry]), HKEY_LOCAL_MACHINE, is probably the most important as it contains all the information about the local system [*i.e.*, computer].”). Indeed, a user of a computer (*i.e.*, server) using a Windows OS can learn the hostname of its computer by writing the “hostname”

command in the Command Prompt. This hostname would be available, or otherwise accessible, also on the above-mentioned Windows registry. Thus, a POSITA would understand that McMillan's uniquely identifiable virtual environments running on McMillan's servers would have their own hostname parameter under their "VirtualEnvs" directory, which identifies the container.

126. Further, as previously discussed, McMillan discloses storing the virtual OS environments (containers) in a central location (memory) that can be accessed by multiple computers. See Ex-1005, 4:57–65. In order to facilitate this, a network configuration must be implemented to connect the multiple computers (*i.e.*, servers), including the central location, which would be located on a central server. In a network configuration, a POSITA would have found it obvious that the unique identity needs to include some mechanism for a computer seeking access to any given container to locate and communicate with the central server on which the container stored. For example, a POSITA would have found it obvious to incorporate a network, such as one taught by Gaines, that connects McMillan's multiple computers (*i.e.*, servers) to a central location. Ex-1007, Fig. 3A (depicting several host servers connected to a network). In any such network, the machines on the network would have addresses that identify each respective individual machine, since peer-to-peer network communication requires that each peer to be able to know which peer it is communicating with. Thus, any container on the central server

(which is itself a peer within the network) would have an associated network address. One popular type of network at the time of McMillan was the IP network, which uses IP addresses to identify devices and other resources on the network. It would have been obvious to a POSITA to use this well-known network in McMillan, especially given the prevalence of TCP/IP stack protocols at the time of McMillan (which use IP addresses), so that McMillan's uniquely identifiable virtual environments (*i.e.*, containers) could be located on the network (*i.e.*, on the central server connected to the network) and retrieved by a computer looking to use the virtual environment (*i.e.*, container). Ex-1009 at 556 ("Since most distributed systems use the Internet as a base, the key protocols these systems use are the two major Internet protocols: IP and TCP."), ("These are called IP addresses and form the basis of routing in the Internet."). Indeed, Gaines expressly references use of protocols (Telnet and FTP) that use IP addresses. Ex-1007, 6:8–21.

127. For the reasons articulated above, a POSITA would have found it obvious to incorporate Tormasov's express teachings of using IP addresses to uniquely identify each of its virtual environment for use by a plurality of user computers over a network, as indicated by Figure 1 below, with McMillan's system.



Ex-1009, Fig. 1 (annotated), [0024] (“each virtual computing environment is the actual remote computer with the network address in which the end user can perform all actions....”). First, a POSITA would have found it obvious to incorporate a network using IP addresses, such as one taught by Tormasov, that connects McMillan’s computers (*i.e.*, servers) to a central location, such as the hardware node taught by Tormasov, because a computer seeking access to any given container would need to be able to communicate with the central server on which this container stored. Second, a POSITA would have found it obvious to incorporate the use of IP

addresses as an identification mechanism for virtual environments, as taught by Tormasov, so that McMillan's computers (*i.e.*, servers) are able to communicate with the central server and locate and access any particular virtual environment.

128. A POSITA would have understood that applying Gaines' teaching of a network, and Tormasov's teaching of unique identities including IP addresses, to McMillan would be a mere combination of prior art elements according to known methods to yield predictable results (*i.e.*, unique identification of McMillan's containers—and McMillan's central server in which they are stored—to facilitate improved access to the containers on the central server by multiple other servers on any given network using IP addresses). Combining Tormasov's teachings with McMillan's system would also amount to the use of a known technique (*i.e.*, a networked environment and unique identities in the form of IP addresses) to improve similar devices (*i.e.*, systems comprising virtual environments that run on multiple computers) in the same way (*i.e.*, facilitating identification of and access to virtual environments stored on a central server by multiple other servers on a network). Therefore, McMillan in combination with Gaines and/or Tormasov renders obvious assigning each of McMillan's containers a unique identity comprising an IP address.

129. Further, in addition to IP addresses, in my opinion, a POSITA would have understood that MAC addresses were well-known address types used to identify particular devices when connected within a local network and to facilitate

communication amongst these devices. MAC addresses are used in many networking technologies, including Ethernet, Wi-Fi, and Bluetooth, in order to uniquely identify each device on a network and facilitate communication amongst these devices. Ex-1019, 1:6:2:11 (“Maintaining unique MAC and IP addresses for each node is critical for proper functioning of the network.”). For the same reasons discussed above with respect to the use of IP addresses to uniquely identify each virtual environment (*i.e.*, container), a POSITA would have found it obvious and beneficial to associate a MAC address for each of McMillan’s containers as part of the unique identities of the containers stored on the central server (*i.e.*, because the use of such identification mechanisms allows any of the multiple servers on a given network to locate and access the containers). In fact, IP packets typically contain both an IP address and MAC address for the destination device. Ex-1019, 1:59-2:11. In my opinion, a POSITA would have found it obvious to include both an IP and MAC address as part of the unique identity for each virtual environment (*i.e.*, container).

4. **Claim 9: “A method as defined in claim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.”**

130. As I previously discussed, claim 2 is obvious. In my opinion, claim 9 is also obvious.

131. McMillan teaches that “[t]he applications running within the virtual OS environments still share the base OS attributes such as networking information, user login rights, services, hardware information, and the Windows clipboard.” Ex-1005, 3:1–4. The most basic of networking information includes network bandwidth, which tells a server and an application running within a virtual OS environment (*i.e.*, container) on that server whether the application is able to communicate over the network (*i.e.*, whether there is sufficient network bandwidth to do so).

132. Further, as discussed above, each of McMillan’s containers includes a file system and registry; this information provides disk allocation information. In other words, McMillan’s virtual OS environments (*i.e.*, containers) run on a single base OS, specifically the base OS of any given host computer. Ex-1005, 1:12–14, Fig. 1. As a result, a POSITA would know that hardware allocation, such as disk allocation, is necessary for the virtual OS environment to operate on the host computer, especially given that by McMillan’s own teaching, “each virtual OS environment appears to be an independent OS for the applications running within it.” Ex-1005, 2:14-15. A POSITA would further understand that while each virtual OS environment believes it has its own physical memory (*i.e.*, disk storage), because the virtual OS environments run on a single base OS of a given host server, the virtual OS environment uses the same physical memory as any other virtual OS environment running on that host server, and as such, each virtual environment

requires an allocation of disk storage.

133. Indeed, McMillan teaches that its virtual OS environments can be loaded onto multiple computers from a central location. As discussed above, when loading virtual OS environments onto any given host computer (*i.e.*, server), a POSITA would have found it obvious, and in fact necessary, to associate a portion of CPU memory, network bandwidth, and/or disk allocation of the host server to each virtual environment prior to application execution. Otherwise, the virtual OS environment, and the applications located therein, could not run on a given computer system because it would not have the system resources necessary to run applications.

134. Gaines likewise explains that applications running in virtual environments use system resources by teaching constraints on resources to be made available to any given application. *See* Ex-1007, 13:52–57 (“The owner or administrator of the server host computer 101 details the constraints on resources the server host computer 101 will make available to virtual applications 143, such as limits on processor time or processor loading, processor memory space, or file system space.”). Therefore, because Gaines’ virtual applications use system resources such as CPU memory and disk space, a POSITA would further understand and find it obvious associate CPU memory, disk allocation, and other system resources with virtual environments running on host computers prior to application execution, because otherwise, the application could not run on a given system.

5. Claim 10: “A method as defined in claim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof.”

135. As previously discussed, claim 2 is obvious. In my opinion, claim 10 is also obvious.

136. McMillan teaches that applications executing within each virtual OS environment (*i.e.*, container) are independent, lacking the ability to access the underlying base OS system files or system files in other virtual OS environments. McMillan teaches that “[a] change made in one environment does not affect the main OS or any other environment” and “[t]he file system and registry information for each environment is independent of the base OS and other environments.” Ex-1005, 2:57–58, 2:60–62; *see also id.*, Cls. 3, 4, 6. Thus, applications running in any given virtual environment does not have access to the base OS or other virtual environments when executing.

137. McMillan also teaches that applications in a virtual OS environment access the virtual OS file system or registry rather than the base OS file system and registry. Ex-1005, 3:11–16, 4:42–45. Specifically, McMillan teaches that any attempts by an application in a virtual OS environment to access the base file system and registry either directly or indirectly results in a redirection such that the applications ultimately access the file system and registry of the virtual OS

environment. *Id.* McMillan further explains that an application running under a specific virtual OS environment “can only make changes to the virtual OS,” “can only place files into a location that is under the virtual OS environment itself,” and “cannot change any file system or registry information in the base OS file system.” Ex-1005, 3:59–4:16; *see also id.*, Figs. 1-2, 4:4-7 (“[A]ny applications that are run from under the virtual OS environment file system are always redirected into that virtual OS environment.”). Therefore, a POSITA would understand that applications running within McMillan’s virtual OS environments execute within their own respective virtual OS environments and do not have access to files in other virtual OS environments.

B. Ground II: Schaefer Alone or in View of Gaines and/or Tormasov

138. Schaefer alone or in combination with Gaines and/or Tormasov renders claims 1, 2, 6, 9, and 10 obvious.

1. Claim 1

- a. **1 [Pre.a]: “In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor,”**

139. In my opinion, 1[Pre.a] is obvious.

140. **First**, Schaefer, alone or in combination with Gaines, renders obvious “a system having a plurality of servers with operating systems that differ.”

141. For example, Schaefer discloses a networked software environment including one or more applications to be run with at least one client computer, an operating system abstraction and protection layer provided by an “OS Guard” component, and an operating system. Ex-1008, [0013], [0019], [0042], Figs. 1–2, 4. This computer is a server because it provides a service to the users of the computer by executing an application on behalf of the users. Ex-1001, 2:13-15 (“This invention provides a solution whereby a plurality of services can conveniently be installed on one or more servers.”); *see also* Ex-1008, [0004] (“The present invention provides a system for creating an application software environment without changing an operating system of a client computer. . . . a virtual environment in which an application may run is provided[.]”). Schaefer expressly indicates that it discloses a system “whereby a virtual environment in which an application may run is provided” to a “client computer.” Ex-1008, [0004].

142. A POSITA would also understand and find obvious that multiple of Schaefer’s client computers (servers) may be deployed on Schaefer’s network because Schaefer is expressly designed to work in a networked environment. For example, Schaefer teaches that “[t]he present invention is particularly useful for applications delivered over a network....” Ex-1008, [0042]; *see also id.*, [0003] (describing in the “Background of t[h]e Invention” how the claimed invention resolves issues in the prior art with respect to “environments where an application is

delivered via a network”). Thus, it would be clear and obvious to a POSITA that Schaefer is directed towards the ability to deliver applications to multiple servers connected on a network and to then facilitate the running of those applications on any given server on that network.

143. Schaefer also expressly discloses what it refers to as an OS Guard and defines the OS Guard as “a layer between a running application and the operating system of a target computer or client computer that provides a virtual environment in which an application may run” and can be applied to computers of differing operating systems, and thus multiple computers. *See, e.g.*, Ex-1008, [0013], [0005] (discussing Windows embodiment), [0026] (discussing application to Windows, Macintosh, and UNIX), [0032] (discussing application to Windows, Macintosh, and UNIX). Schaefer also explicitly contemplates the application of its teachings to various operating systems, explaining that “[i]n the present invention, it is recognized that there are several components of the invention, the behavior or presence of which is different on alternate operating systems.” Ex-1008, [0050]. Indeed, Schaefer’s OS Guard plays an important role in enabling applications to be deployed onto any of the multiple computers within Schaefer’s networked environment, where each computer uses a different underlying base OS distribution, specifically given its role in provisioning system files (*e.g.*, configuration files and/or shared library files), device drivers, and other resources and components to the

applications running in virtual environments. *See, e.g.*, Ex-1008, [0021]-[0054].

144. In view of the foregoing, it would at least have been obvious to deploy multiple of Schaefer's client computers (servers) in a networked configuration where different operating systems are used. Further, the use of networked computers with different operating systems was a well-known and common deployment in practice, and would increase the usability, versatility, and flexibility of Schaefer's networked environment, because it would not limit its use to a single OS.

145. Gaines also renders obvious the use of multiple computers and operating systems. As I previously discussed above for McMillan, Gaines teaches a networked execution environment comprising multiple different computers (*i.e.*, servers), each having a different operating system, where applications can execute in different virtual environments on any of the servers within the networked environment.

146. This use of different operating systems increases the usability, flexibility, and versatility of Gaines' system, because Gaines' system is not locked into any particular operating system, which in turn allows applications to run in virtual OS environments on servers having any one of multiple underlying base OSs. Thus, POSITA would be motivated to use multiple operating systems with Schaefer to achieve these same benefits for all of the same reasons I previously discussed for McMillan.

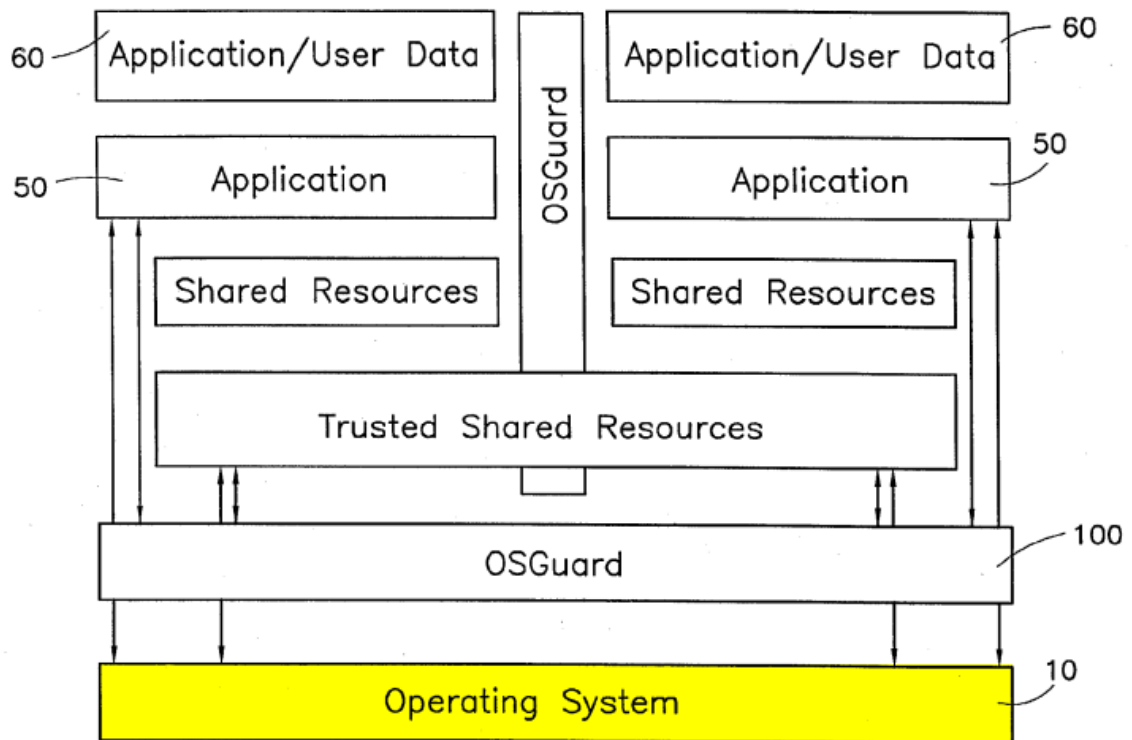
147. **Second**, in my opinion, Schaefer’s client computers (*i.e.*, servers) with different operating systems, when deployed as part of a networked environment, operate in disparate computing environments under both parties’ constructions of “disparate computing environments.” Schaefer’s computers (*i.e.*, servers) meet PO’s construction of “environments run by standalone computers,” because each of Schaefer’s computers is separate from the others. Ex-1001, 2:17-19. They are distinct, separate, individual servers. Schaefer’s computers also meet Petitioner’s proposed construction of “[e]nvironments where computers are stand-alone or where there are plural computers and where they are unrelated,” for this same reason, and because Schaefer’s computers would be separate standalone computers connected to a network, with no relationship among them other than that the computers are peers with one another by virtue of connecting to the same network.

148. **Third**, the use of a processor and an operating system including a kernel [and] a set of associated local system files compatible with the processor is also obvious.

149. As I previously discussed for McMillan, a POSITA would understand and find obvious that a computer includes a physical computer processor—the presence of a processor is the defining feature that renders a computer a computer. *See supra* Section IX.A.1.a. That section is incorporated by reference as if fully set forth herein. Schaefer, just like McMillan, makes multiple references to “running”

a software application, which a POSITA would understand to necessarily implicate the presence of a processor—*i.e.*, processors run software such as applications.

150. Further, each of Schaefer’s computers contains an operating system, including as illustrated in Figure 1 below. *See* Ex-1008, [0012] (“Disposed between the application 50 and operating system 10 ...”), [0063] (“the operating system guard 100 serves as an abstraction layer ... between the application software 50 and the actual operating system 10 of the client computer.”). A POSITA would understand and find obvious that an operating system requires a processor in order to run (*i.e.*, the operating system is software, and processors run software). Further, an operating system is responsible for managing hardware resources like CPU utilization, which would be required to run any application on a computer (*i.e.*, server) in Schaefer’s networked environment. Thus, a POSITA would understand that a processor is present on Schaefer’s computers, which run applications delivered via a network.



Ex-1008, Fig. 1 (annotated).

151. Further, POSITA would understand and find obvious that any OS has a kernel because the kernel is the part of the OS that controls interactions between a computer’s hardware and the software executing on a computer. Ex-1009 at 461, Fig. 11–7. Specifically, it would be well-known to a POSITA that the kernel performs a variety of important functions relating to process management, inter-process communications, memory management, device (hardware) management, file system management, resource management, and security and access protocols. *See, e.g.*, Ex-1008, [0022] (describing the subsystems of Schaefer’s OS Guard, which relate to core OS functions); Ex-1009, 778 (“The kernel is a traditional kernel

in the sense that it handles management, memory management, file systems, and so on.”), 771-852 (discussing kernel functions and processes in a Window OS distribution); *see also, e.g.*, Ex-1009, 5-6 (discussing the operating system as a “resource manager”), 44-56 (discussing system calls, including Win32 API calls), Fig. 1-17.

152. **Fourth**, Schaefer discloses that a base OS has associated system files, which are present in all operating systems and were well-known to a POSITA. *See* Ex-1009 at 38. These include services, shared libraries (*i.e.*, DLLs), configuration files, and device drivers. *See* Ex-1009, 771-852 (discussing Windows 2000 file systems, system files, including DLLs and device drivers, APIs, and kernel objects, which represent system resources). Indeed, Schaefer explains that Windows-based OSs provide specific system files (*e.g.*, services) to applications. Ex-1008, Fig. 1, [0005]; [0025]-[0054] (discussing system files, including shared library files, device drivers, and other resources and components found in OSs such as Windows and Unix).

153. Further, Schaefer explicitly reserves an entire section dedicated towards “Shared Objects,” which includes discussion regarding shared library files across Windows and Unix systems. Ex-1008, [0031]-[0042]. Thus, a POSITA would understand that Schaefer’s system files include dynamically-linked libraries (“DLLs”), or shared libraries, not only because of Schaefer’s express teachings, but

also because a POSITA would also understand that DLLs provide applications with services to access portions of any given file system and perform other necessary functions. *See, e.g.*, Ex-1008, Abstract (“These components cover the protection semantics required by .DLLs and other shared library code”), [0013] (“For example, if an application attempts to change the version of a shared object like MSVCRT.DLL, this change is localized to the application and the code resident on the client computer is left untouched.”), [0032] (“On the Windows platform Dynamic Link Libraries, DLLs, are often shared within and across applications.”).

154. A POSITA would also recognize that Schaefer’s express disclosure of the use of system calls and APIs, which are implicated in the process of accessing the file system and registry, teach the presence of system files within the base OS file system and registry, as system calls and APIs are integral to the process by which an application would access system files, including services, in the file system. *See generally* Ex-1008.

- b. 1 [Pre.b]: “a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment,”**

155. In my opinion, 1[Pre.b] is obvious.

156. **First**, Schaefer teaches an OS Guard which “defines a layer between a running application and the operating system of a target computer or client computer that provides a virtual environment in which an application may run.” Ex-1008,

[0013]; *see also id.*, [0007], [0019] (“each application is able to run in a private context within the system.”), Fig. 2, [0040] (explaining that a virtual environment has a virtual file system and virtual registry). This is a “secure environment” as claimed. Indeed, Schaefer’s OS Guard plays a critical role in securing the virtual environment from the running base OS environment and other virtual environments. *See, e.g.*, Ex-1008, Fig. 1, [0012], [0013]-[0015], [0022]-[0063] (discussing the OS Guard’s role in provisioning the Schaefer’s private contexts); *see also, e.g., id.*, [0038] (“In general, unless specifically configured as such, shared objects should be loaded private to ensure conflict prevention. . . . if the system crashes, it is required to recover from this crash to a clean slate, not having overwritten or modified the underlying operating system.”), [0041] (“If an application tries to create a new file or open an existing file for writing on a user’s local drive, the Operating System Guard must ensure that the file is actually created or modified in the redirected location. If the application is reloaded at a later time, this file mapping must be reloaded into the Operating System Guard virtual environment. When the request is to modify an existing file, which resides on a user’s local drive, the Operating System Guard must copy the file in question to the redirection point before continuing with the request. The redirected files may not be of the same name as the original file to ensure safe mapping of file paths. In the preferred embodiment, MI files are handled in this way to offer maximum System Security while allowing

maximum application compatibility.”), [0052] (“[T]he Operating System Guard can use its File subsystem to avoid placing the actual font file in the running system.”).

157. Further, Schaefer explicitly describes examples of the applications that can run in the virtual environments such as Adobe Photoshop, Microsoft Office, and Microsoft Word. *See id.*, [0014], [0020]. Schaefer explains that “many applications use Microsoft Word as an engine for doing Mail Merge or document creation functionality.” Ex-1008, [0020]. Thus, Schaefer teaches providing servers with applications related to a service. Further, a POSITA would understand that these applications, beyond those explicitly referenced by Schaefer, can be related to multiple services, such as an advertising service or banking service as explicitly taught by Gaines. *See* Ex-1007, 11:47–52. These applications not only relate to service, but are also “secure” because Schaefer’s OS Guard “prevents information that may exist on the client/users machine from interfering with or modifying the behavior of an application” and by allowing “each application ... to run in a private context within the system....” Ex-1008, [0007], [0012], [0015], [0019], [0023] (discussing the “process manager 120 provided by the Operating System Guard”), [0063] (“The Operating System Guard 100 keeps the two instances of the application 50 from interfering with one another.”), Fig. 1 (depicting an OS Guard between private contexts and between private contexts and the operating system).

158. **Second**, Schaefer’s virtual environments, which are also referred to as

“private contexts,” provide secure environments for each application to execute. For example, Schaefer depicts its private contexts in Figure 2, where there are two separate applications or different versions of the same application (52, 54) that execute independently and utilize different sets of system files (*i.e.*, system files unique to each private context), thereby providing secure environments, as just as the ’814 Patent teaches. Ex-1008, Fig. 2, [0007], [0013]-[0015] (teaching its claimed virtual environment “presents an environment to a running application that appears to be an installation environment without performing an installation”), [0019]-[0020]; Ex-1001, 7:4–10 (explaining that applications executing in a secure environment “is made possible by associating applications with secure containers” such that “[a]pplications are segregated and execute with their own copies of files and with a unique identity”).

159. Thus, in Schaefer, applications are able to execute within these private contexts because within each private context exists a virtual file system and registry, and applications within each private context execute independently and access files from the virtual file system and registry for that particular private context. *Id.* at [0012], [0019], [0040] (“The present invention provides a virtual file system much like the virtual registry described above.”), [0025]-[0054] (discussing system files, including shared library files, and other resources and components found in OSs such as Windows and Unix, to be provided to an application running in a private

context), Fig. 1, Fig. 2 (showing “system files” within “private contexts”), Fig. 4 (illustrating virtual environment manager with “files” and “configuration”). In my opinion, a POSITA would understand from Schaefer’s disclosure that an application cannot execute in another private context other than the one it was assigned.

c. 1 [Pre.c]: “wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:”

160. In my opinion, 1[Pre.c] is obvious.

161. **First**, Schaefer discloses or renders obvious that “the applications each include an object executable ... for performing a task related to the service.”

162. Schaefer describes examples of the applications that can run in the virtual environments, such as Adobe Photoshop, Microsoft Office, and Microsoft Word. *See* Ex-1008, [0014], [0020]. Schaefer explains that “many applications use Microsoft Word as an engine for doing Mail Merge or document creation functionality.” Ex-1008, [0020].

163. Schaefer also teaches that the virtual environments can provide private contexts for a given application to run. *See id.*, [0019] (“each application is able to run in a private context within the system.”), Figs. 1-2. Schaefer explains that “many applications use Microsoft Word as an engine for doing Mail Merge or document creation functionality.” Ex-1008, [0020]. Thus, Microsoft Word can be used for performing a task (*i.e.*, merging mail or creating a document) for a given

service. Schaefer also discusses font directories and associated APIs in the context of a Windows OS. Ex-1008, [0049]-[0052]. A POSITA would understand that in the Windows context, applications such as least Notepad™ text editor, Wordpad™ word processor, or Internet Explorer® internet browser are for performing a task related to the service such as font formatting. For example, a service provided by Notepad™ text editor or Wordpad™ word processor can include editing a document, and specific tasks related to this service can include typing words, font formatting, and text manipulation, among many others.

164. Further, a POSITA would understand that, as another example, Microsoft Excel can be used for performing a task (*i.e.*, updating the data in a row) for a given service, such as an advertising service or banking service as explicitly taught by Gaines. *See* Ex-1007, 11:47–52; A POSITA would understand that the applications running in a given virtual environment would include an object executable, as Schaefer explicitly teaches that the virtual environments are provided for applications to run, and that a running application requires executable object code. *See* Ex-1008, [0004], [0013–0014], [0019]; Ex-1009 at 814, 267. It would be well-known to a POSITA that the purpose of an application executables (*i.e.*, execution files) is to provide the operating system with instructions for running an application program on the system. Without such instructions, the computer would not be able to run the given application. Further, executable files are necessary to

coordinate interactions between the application program and the operating system, especially at run-time, where the operating system often provisions resources to the application for it to run effectively. Ex-1009, 21-23 (discussing program instructions and how the OS and CPU work together to execute such instructions), 44-56 (discussing system calls and how the OS and CPU work together to carry out system calls from an application); *see also id.*, 5-6, Fig. 1-17.

165. **Second**, a POSITA would find it obvious that the applications running in the virtual environments include an object executable “by at least some of the different operating systems.” *See supra* Claim 1[Pre.a]. In order to illustrate this, consider a system of 5 servers, including one server running Unix, two servers running Windows NT, and another running Windows 2000. As described above with regard to McMillan, any application written for one of these Windows operating systems would be executable on a different instance of that same operating system, or on the other Windows operating systems because they both use the Windows API. Schaefer even references Win32 API, which was well known to a POSITA that the Win32 API “allows programs to run on almost all versions Windows,” including Windows NT and Windows 2000. Ex-1008, [0005], [0012], [0052], Cl. 5; Ex-1009 at Fig. 11-4 (“The Win32 API allows programs to run on almost all versions of Windows.”).

166. Further, it would also be obvious to incorporate Gaines’ teaching of its

“virtual application interface,” “resource filters,” and “language engine” with Schaefer, allowing the same container to be used on different operating systems such as Windows and Unix for the same reasons discussed in Section IX.A.1.c with regard to McMillan. *See* Ex-1008, Abstract, Figs. 1, 3A, 4:23-28 (“server program [which provides the virtual host operating system and] includes an interpreter for the (interpreted) programming language, a process control subsystem, and a virtual file subsystem.”), 6:28-9:35 (discussing the “virtual application interface” and “resource filters”), 9:36-10-7 (discussing the “language engine” and the interplay between the “language engine” and the “virtual application interface”); *see also supra* Section IX.A.1.c.

- d. **1[a]: “storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers;”**

167. In my opinion, limitation 1[a] is obvious.

168. **First**, Schaefer’s multiple computer deployment renders obvious “a plurality of secure containers of application software,” where “each container compris[es] one or more of the executable applications and a set of associated system files required to execute the one or more applications.” Schaefer’s virtual environments and OS Guard provide secure private contexts (*i.e.*, secure containers)

for each application to run in. Ex-1008, [0012], [0013]-[0015], [0019] (“each application is able to run in a private context within the system.”), [0040], Figs. 1, 2, 4; *see also* [0022]-[0063] (discussing the OS Guard’s role in provisioning the Schaefer’s private contexts). Schaefer explicitly teaches that when an application runs in a private context, it “has its own private view of what the system looks like and its behavior” so the applications in private contexts “appear to have separate or differing copies of system services, configuration and data.” *Id.*, [0019]; *see also id.*, [0017] (“[B]y implementing the present invention, two instances of the application may now run on the same client computer, each connecting to a different database.”), [0025]-[0054] (discussing system files, including shared library files, device drivers, and other resources and components found in OSs such as Windows and Unix that are provided to applications running in private contexts), Fig. 2 (showing “system files” within “private contexts”), Fig. 4 (illustrating virtual environment manager with “files” and “configuration”).

169. Indeed, Schaefer explains that “the present invention can [] add [files] to the virtual environment or files to redirect to another within the virtual environment,” which indicates files being present in each private context. Ex-1008, [0040], [0042]; *see also id.*, [0026]-[0027] (“[A] virtual Windows Registry component [[]] will provide a full function registry to an application, but prevent modification to the underlying system registry.”), [0038] (“shared objects should be

loaded privately to ensure conflict prevention.”), [0044] (“[T]he Operating System Guard will provide the capability of dynamically, and as possible privately, adding and removing [device drivers or other operating system level software] to an application’s virtual environment.”).

170. Further, Schaefer teaches that the OS Guard provides a virtual environment manager, which creates the virtual environment (*i.e.*, private contexts) through its provisioning of virtual files, configuration data, shared objects, and other resources and components, and a virtual file system manager:

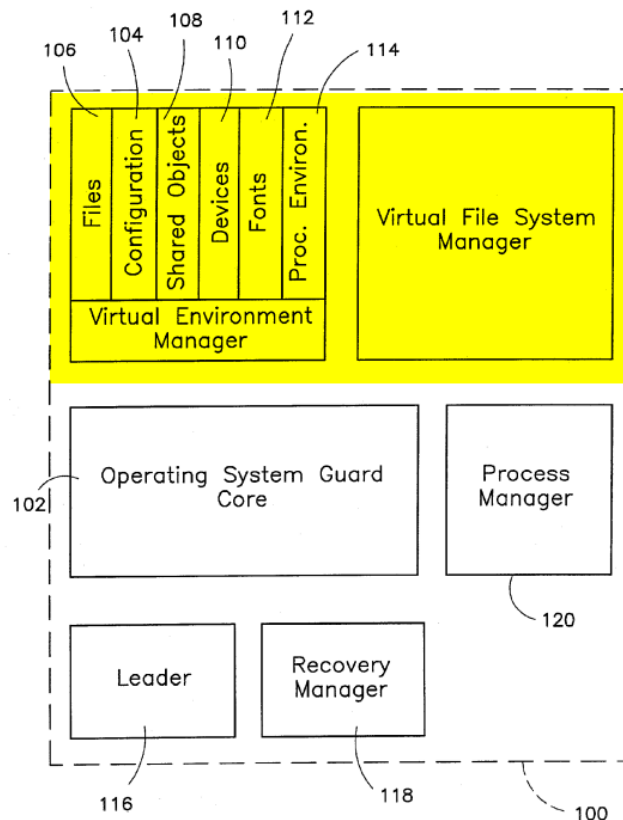


Fig. 4

Ex-1008, Fig. 4 (annotated). Because Schaefer’s OS Guard creates and manages

virtual environments and their virtual file systems by managing the provision of differing copies of system files, configuration data, shared objects, device drivers, and other resources into the virtual environment, a POSITA would understand that Schaefer's private contexts are "containers," which are "[a]n aggregate of files required to successfully execute a set of software applications on a computing platform." Ex-1001, 2:23–25; Ex-1008, [0040], [0042]; *see also* Ex-1008, [0026]–[0027] ("[A] virtual Windows Registry component [[]] will provide a full function registry to an application, but prevent modification to the underlying system registry."), [0038] ("shared objects should be loaded privately to ensure conflict prevention."), [0044] ("[T]he Operating System Guard will provide the capability of dynamically, and as possible privately, adding and removing [device drivers or other operating system level software] to an application's virtual environment."), Fig. 2, Fig. 4.

171. Schaefer also teaches that its virtual environments provide a private contexts for each application to run, which a POSITA would understand includes executable applications (*e.g.*, Microsoft Word or Excel), since executables are necessary to provide the operating system with instructions for running an application program on the system. See Ex-1008, [0013]–[0014], [0019]–[0020]; *see also supra* Section IX.B.1.c. Therefore, because each of Schaefer's private contexts includes executable applications and private copies of "system services,"

“configuration [files],” “shared objects,” and other resources and components, a POSITA would understand that Schaefer’s private contexts are “containers” because they are an “[a]n aggregate of files required to successfully execute a set of software applications on a computing platform” and are thus a container. Ex-1001, 2:23–25.

172. A POSITA would have further understood and found obvious that Schaefer’s virtual environments (*i.e.*, private contexts), which are provided by Schaefer’s OS Guard, constitute “secure containers of application software” under the parties’ agreed construction: “[a] container environment where each application set appears to have individual control of some critical system resources and/or where data within each application set is insulated from effects of other application sets,” wherein “each container for use on a server is mutually exclusive of the other containers, such that read/write files within a container cannot be shared with other containers.” Specifically, Schaefer teaches that the virtual environments (*i.e.*, containers) provide secure environments for applications to run, which localize all changes to the virtual environment, and appears to have its own control of resources, files, and data, as opposed to making actual changes to the underlying OS or other containers. For example, in Schaefer, applications running in a private context cannot modify the underlying operating system settings of the host computer, as any attempts to do so result in such modifications only being made in the virtual environment. Ex-1008, [0013]. Therefore, the applications have individual control

over at least some of the critical system resources within the private context. Ex-1008, [0013] (explaining that “[i]f an application attempts to change underlying operating system settings of a client computer, such settings are protected and only ‘made’ in the virtual environment” and “localized to the application and the code resident on the client computer is left untouched.”).

173. As another example, Schaefer explains that although “by implementing the present invention, two instances of the application may [] run on the same client computer,” “each connect[s] to a different database,” and the OS Guard “keeps the two instances of the application 50 from interfering with one another”. See Ex-1008, [0017], [0063], Figs. 1-2. Schaefer further explains that “many operating systems 10 provide fault domains to protect applications 50 from affecting each other when run . . . [a]n operating system abstraction and protection layer 100 [*i.e.*, Schaefer’s OS Guard as depicted in Figure 1], will provide an additional, programmatically controlled barrier between applications 50 to remove most application level interactions.” Ex. 1008, [0012]. Figure 1 in Schaefer also shows that the OS Guard protects “Application/User Data” and an “Application” from other application sets running in another private context. Therefore, applications running in Schaefer’s private contexts and their data are insulated from the effects of other application sets.

174. Thus, a POSITA would understand that Schaefer’s OS Guard provides for virtual environments or private contexts (*i.e.*, containers), wherein each

application running in a private context is insulated from applications running in other private contexts, as indicated by Figures 1, and read/write files within a container are not shared with other containers. *See* Ex-1008, [0007], [0012], [0013]-[0015] (teaching that its virtual environment “prevents a running application from making changes to the client computer,” “presents an environment to a running application that appears to be an installation environment without performing an installation,” and “prevents information that may exist on the client/users machine from interfering with or modifying the behavior of an application.”), [0017], [0019], [0026] (“[A] virtual Windows Registry component [[]] will provide a full function registry to an application, but prevent modification to the underlying system registry.”), [0038] (“shared objects should be loaded private to ensure conflict prevention.”), [0040], [0041] (“If an application tries to create a new file or open an existing file for writing on a user’s local drive, the Operating System Guard must ensure that the file is actually created or modified in the redirected location. If the application is reloaded at a later time, this file mapping must be reloaded into the Operating System Guard virtual environment. When the request is to modify an existing file, which resides on a user’s local drive, the Operating System Guard must copy the file in question to the redirection point before continuing with the request. The redirected files may not be of the same name as the original file to ensure safe mapping of file paths. In the preferred embodiment, MI files are handled in this way

to offer maximum system security while allowing maximum application compatibility.”), [0044] (“[T]he Operating System Guard will provide the capability of dynamically, and as possible privately, adding and removing [device drivers or other operating system level software] to an application’s virtual environment.”), [0063], Figs. 1-2.

175. **Second**, Schaefer’s multiple-computer embodiment renders obvious “storing in memory accessible to at least some of the servers” the plurality of secure containers (*i.e.*, virtual environments/private contexts provided by the OS Guard). A POSITA would have found it obvious that because multiple of Schaefer’s computers can be networked in order to deliver applications to any one of those computers, any computer can access another container (*i.e.*, virtual environment/private context), which would be stored in memory on the network, such as on one or more computers connected to the network. Ex-1008, [0042]; *see also id.*, [0003] (describing in the “Background of t[h]e Invention” how the claimed invention resolves issues in the prior art with respect to “environments where an application is delivered via a network”).

176. Further, a POSITA would have found it obvious that any of Schaefer’s computers could transfer and/or download virtual environments (private contexts) to any other. For example, Schaefer explains that its OS Guard includes a component “loader subsystem 116” (red) which “is used to allow virtual environments to be

transferred into and out of the running system.” Ex-1008, [0024].

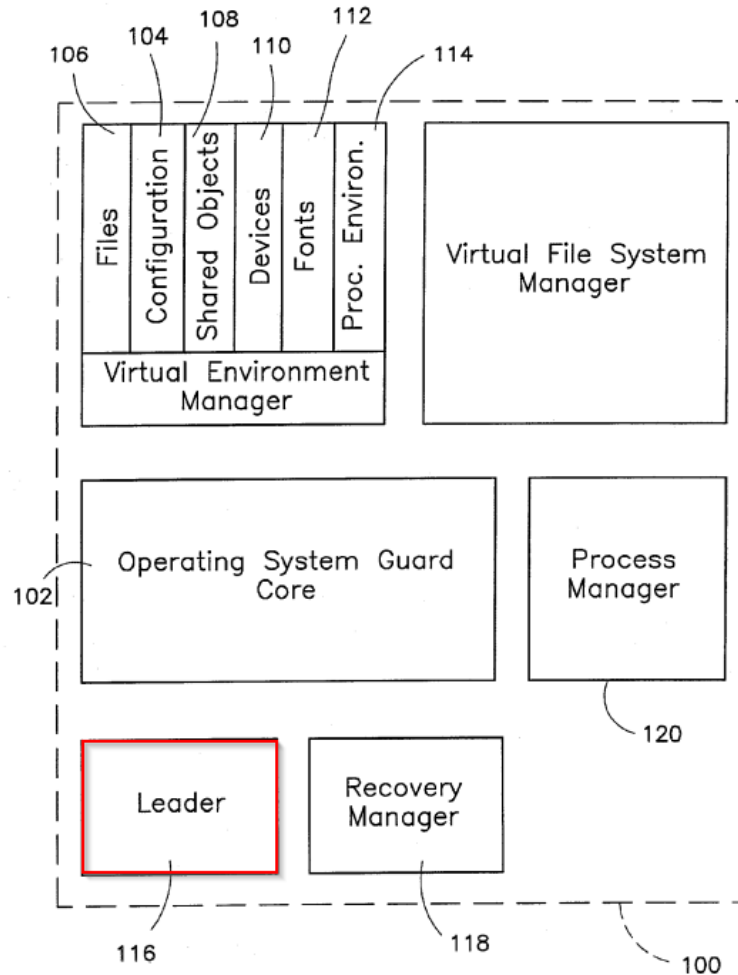


Fig. 4

Ex-1008, Fig. 4 (annotated). Based on Schaefer’s specification, “leader” in Figure 4 is a typographical error and should read “loader,” and was corrected in the issued U.S. Patent No. 7,028,305 (Ex-1020).

177. A POSITA would thus have understood and found it obvious that Schaefer’s virtual environments (*i.e.*, private contexts) created by Schaefer’s OS Guard (*i.e.*, secure containers of application software) would thus be stored in

memory, from which any virtual environment (*i.e.*, private contexts) could be loaded and unloaded into a given networked computer via Schaefer's OS Guard's loader subsystem. As Schaefer explains, "[e]ach of the Virtualization Subsystems [which includes: "core 102, configuration manager 104, file manager 106, shared object manager 108, device manager 110, font manager 112, [] process environment manager 114, [] and recovery manager 118"] is capable of serializing its configuration for the loader 116, and retrieving it through the reverse process. In addition, the loader 116 is capable of staged loading/unloading and combining the results of individual stages into one single environment description." Ex. 1008, [0022], [0024].

178. This would also be obvious in view of Gaines. Gaines teaches a distributed execution environment, which similarly provides the ability to transfer programs across computers over the network. A POSITA would find it obvious to apply that teaching (*i.e.*, the ability to transfer programs among computers connected to a network) to Schaefer's networked environment in the same manner because it would amount to a simple substitution of known prior art elements and is consistent with the express teachings of Schaefer. *See* Ex-1007, 13:40–44, Fig. 3A. It would have been further obvious to incorporate this teaching into Schaefer, given the similarities between Schaefer's and Gaines' architectures and given that the ability to transfer applications between computers in a networked environment enhances

the operation of the overall system by making applications available for retrieval and use by multiple computers in the network. This combination would amount to the use of a known technique (*i.e.*, applying a peer-to-peer network environment such that computers can share files, applications, and data amongst each other) to improve similar systems (*i.e.*, allowing multiple computers on a network to retrieve and use applications stored on a central server) in the same way.

179. **Third**, Schaefer's multiple-computer networked environment renders obvious that the secure containers are "for use with a local kernel residing permanently on one of the servers." Schaefer teaches that the virtual environment (*i.e.*, private contexts) provided by the OS Guard (*i.e.*, secure containers of application software), which, as explained previously plays a critical role in securing the virtual environment from the running base OS environment and other virtual environments, are primarily (though not exclusively) for use with a given computer with a base OS. Specifically, Schaefer explains (and as illustrated in Fig. 1) that certain commands pass through the OS Guard "and are then transferred to the Operating System itself." Ex-1008, [0063], Fig. 1 (illustrating the relationship between an operating system and software application). Therefore, Schaefer's virtual environments exist on a host computer having a base OS, which requires a kernel in order to effectively run the computer system (*i.e.*, interactions between the hardware of a computer and the software executing on a computer). Ex-1009 at 461,

Fig. 11-7.

180. In my opinion, a POSITA would understand that the underlying OS (10) in Figure 1 includes the kernel because, again, the kernel is a critical component of the OS that controls interactions between a computer's hardware and the software executing on a computer. Ex-1009 at 461, Fig. 11-7. Thus, the virtual environment (*i.e.*, private contexts) provided by the OS Guard (*i.e.*, secure containers of application software) are for use with the local kernel residing permanently on any given computer on Schaefer's networked environment. Ex-1008, Fig. 1 (annotated). Further, a POSITA would understand that Schaefer's OS Guard comprises subsystems that necessarily requires interfacing with the base OS kernel. Ex-1008, [0022].

e. 1[b]: "wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems,"

181. In my opinion, limitation 1[b] is obvious.

182. This element is disclosed or rendered obvious by Schaefer. Consider a configuration of five servers, all of which are connected to a central server through a network, where one server runs Unix, two servers run Windows NT, and another server runs Windows 2000. In a virtual environment (*i.e.*, private context) designed for operation with a base Windows NT OS, the virtual environment would include the system files compatible with the local kernel of at least the computers running

Windows NT as a underlying base OS distribution. The operating systems on these Windows NT servers are “at least some of the plurality of different operating systems” within the configuration set forth above. Additionally, the set of associated system files would also be compatible with the local kernel of the computers running Windows 2000 as a base OS, given that, as previously explained both Windows NT and Windows 2000 would utilize Win32 API, which “allows program to run on almost all versions of Windows.” *See* Ex-1009 at Fig. 11-4 (“The Win32 API allows programs to run on almost all versions of Windows.”), (illustrating Windows NT and Windows 2000).

183. Further, as I discussed previously with regard to McMillan, Gaines expressly teaches that a “virtual operating system is capable of executing programs in a standardized programming language, which may be either an interpreted or compiled language, to provide the remote user with the ability to run programs that are host-independent.” Ex-1007, 4:13-18. Gaines further teaches a plurality of computers with different operating systems, even identifying a broad range of operating systems that may execute on any given host computer, and that “[t]he host virtual operating system may be programmed for any of these hardware and operating system software.” Ex-1007, 5:1-16, 3:66-4:2 (recognizing the advantage of having “a virtual execution environment for which programs could be written that are capable of execution on different computer hardware and with different

operating systems”).

184. In my opinion, a POSITA would be motivated to apply Gaines’ “language engine” (and “virtual application interface” and “resource filters”) with Schaefer to achieve these same benefits (*i.e.*, increasing the usability, versatility, and flexibility of Schaefer’s system) for all of the same reasons I previously discussed. *See* Ex-1008, Abstract, Figs. 1, 3A, 4:23-28 (“server program [which provides the virtual host operating system and] includes an interpreter for the (interpreted) programming language, a process control subsystem, and a virtual file subsystem.”), 6:28-9:35 (discussing the “virtual application interface” and “resource filters”), 9:36-10:7 (discussing the “language engine” and the interplay between the “language engine” and the “virtual application interface”).

- f. **1[c]: “the containers of application software excluding a kernel, wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server,”**

185. In my opinion, limitation 1[c] is obvious.

186. Schaefer’s private contexts run on top of an operating system where the kernel is located, and therefore excludes a kernel. Figure 1 provided by Schaefer illustrates this, as it shows a single operating system, on top of which exists applications in secure virtual environments provided by the OS Guard. Ex-1008, Fig. 1. The OS Guard servers as a “barrier” between two applications running in two

different private contexts, and more relevant here, a “barrier” between all private contexts and the operating system. *Id.* The private contexts themselves are not suggested to have their own specific operating systems, much less a kernel. *Id.* Further, as previously discussed, Schaefer’s private contexts (including the copies/modified copies of the associated system files) need to be compatible with the various OSs (including local kernel) on which the containers are running, including because Schaefer explicitly discusses that certain commands pass through the OS Guard “and are then transferred to the Operating System itself.” Ex-1008, [0063]. Schaefer’s private contexts would not function as designed if they were not able compatible with the OS of the host computer. In my opinion, a POSITA would understand that because the virtual environment (*i.e.*, private contexts) communicate with the underlying base OS in order to successfully run as secure virtual environments (*i.e.*, private contexts), the virtual environment/private contexts do not include a kernel.

187. Moreover, as previously discussed, Schaefer discloses that each of the virtual environment (*i.e.*, private contexts) provided by the OS Guard (*i.e.*, secure containers of application software) “have separate or differing copies of system services, configuration and data.” Ex-1008, [0019], [0026] (“[A] virtual Windows Registry component [[]] will provide a full function registry to an application, but prevent modification to the underlying system registry.”), [0038] (“shared objects

should be loaded privately to ensure conflict prevention.”), [0040] (“The present invention provides a virtual file system much like the virtual registry described above.”), [0044] (“[T]he Operating System Guard will provide the capability of dynamically, and as possible privately, adding and removing [device drivers or other operating system level software] to an application’s virtual environment.”) [0025]-[0054] (discussing system files, including shared library files, and other resources and components found in OSs such as Windows and Unix that are provided to applications running in private contexts), Fig. 2 (showing “system files” within “private contexts”), Fig. 4 (illustrating virtual environment manager with “files” and “configuration”). Therefore, rather than to use the system files local to the host computer (*i.e.*, in the base OS file system), applications running in Schaefer’s private contexts provided by the OS Guard (*i.e.*, containers) access its own private copies of system files (*i.e.*, in the virtual file system).

188. Thus, in my opinion, a POSITA would understand and find obvious that each container utilizes its own copies of the system files in place of the system files located on the underlying base OS. A POSITA would further understand and find obvious that applications running within each secure virtual environment (*i.e.*, secure containers of application software) use copies of the system files that it needs to run from the virtual file system rather than the system files located on the underlying base OS.

- g. **1[d]: “wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server,”**

189. In my opinion, limitation 1[d] is obvious.

190. As previously discussed, Schaefer’s private contexts (*i.e.*, containers) utilize “separate or differing copies of system services, configuration and data.” Ex-1008, [0019], [0026] (“[A] virtual Windows Registry component [[]] will provide a full function registry to an application, but prevent modification to the underlying system registry.”), [0038] (“shared objects should be loaded privately to ensure conflict prevention.”), [0040] (“The present invention provides a virtual file system much like the virtual registry described above.”), [0044] (“[T]he Operating System Guard will provide the capability of dynamically, and as possible privately, adding and removing [device drivers or other operating system level software] to an application’s virtual environment.”), [0025]-[0054] (discussing system files, including shared library files, and other resources and components found in OSs such as Windows and Unix), Fig. 2 (showing “system files” within “private contexts”), Fig. 4 (illustrating virtual environment manager with “files” and “configuration”).

191. A POSITA would understand that the system files in each of Schaefer’s private contexts are copies or modified copies of the system files than remain on the base OS because the system files (*i.e.*, services, configuration files, shared libraries,

device drivers, etc.) are added into the virtual environments from the base OS file system or one of the subsystems of the OS Guard. *See* Ex-1008, [0014], [0019], [0027], [0035]-[0036], [0040]-[0042], [0044]. In other words, the OS Guard is the layer between the software application executing in a private context and the operating system and is responsible for provisioning files from the local (*i.e.*, base) OS file system into virtual file systems. Thus, although applications running in Schaefer's private contexts use their own private copies of system files accessible through virtual file systems, whether the system files remain on the base OS file system or one of OS Guard's subsystems, the system files remain on the server.

- h. 1[e]: “and wherein the application software cannot be shared between the plurality of secure containers of application software, and wherein each of the containers has a unique root file system that is different from an operating system's root file system.”**

192. In my opinion, limitation 1[e] is obvious.

193. As I previously discussed, Schaefer teaches that the private contexts (containers) provided by the OS Guard provide secure environments such that the applications are constrained to that particular context. *See* Ex-1008, [0007], [0012], [0013]-[0015] (teaching that its virtual environment “prevents a running application from making changes to the client computer,” “presents an environment to a running application that appears to be an installation environment without performing an installation,” and “prevents information that may exist on the client/users machine

from interfering with or modifying the behavior of an application.”), [0017], [0019], [0038] (“In general, unless specifically configured as such, shared objects should be loaded private to ensure conflict prevention.”), [0040], [0041] (“If an application tries to create a new file or open an existing file for writing on a user’s local drive, the Operating System Guard must ensure that the file is actually created or modified in the redirected location. If the application is reloaded at a later time, this file mapping must be reloaded into the Operating System Guard virtual environment. When the request is to modify an existing file, which resides on a user’s local drive, the Operating System Guard must copy the file in question to the redirection point before continuing with the request. The redirected files may not be of the same name as the original file to ensure safe mapping of file paths. In the preferred embodiment, MI files are handled in this way to offer maximum system security while allowing maximum application compatibility.”), [0063], Figs. 1-2. Thus, in my opinion, in Schaefer, the software for a particular application software cannot be shared between the plurality of secure containers, rendering this limitation obvious.

194. Further, in my opinion, a POSITA would have understood and found it obvious that each of the containers has its own unique root file system that is different from the OS’s root file system. Schaefer explicitly teaches that the virtual environment (*i.e.*, private contexts) provide a virtual file system, which is unique and different from the underlying OS’s file system. Ex-1008, [0040]. Schaefer

provides Figure 4, which clearly depicts the “virtual file system” that is encapsulated by the operating system abstraction and protection layer, which separates the virtual environment (*i.e.*, private context) from the base operating system and its base file system. Ex. 1008, Figs. 1, 4.

2. Claim 2: “A method as defined in claim 1, wherein each container has an execution file associated therewith for starting the one or more applications.”

195. As I previously discussed, claim 1 is obvious. In my opinion, claim 2 is also obvious.

196. As I previously discussed, Schaefer teaches that different applications can run in private contexts of the system. *See* Ex-1008, [0017] (“Two instances of the application may now run on the same client computer, each connecting to a different database”), [0019] (“[T]wo separate applications 52, 54, or two instances of the same application [] can be provided private contexts in which they will appear to have separate or differing copies of system services, configuration, and data.”), Cl. 7 (“wherein said operating system abstraction and protection layer manages the integration of multiple instances of an application by recognizing how many instances of an application are running.”); *see also id.* at [0012], [0063], Figs. 1, 2, 4.

197. In my opinion, a POSITA would understand that in order for an application to run, an execution file would be required to start the application within

the private context, since the purpose of an application executables (*i.e.*, execution files) is to provide the operating system with instructions for running an application program on the system, and executable files are necessary to coordinate interactions between the application program and the operating system, especially at run-time, where the operating system often provisions resources to the application for it to run effectively. Compiled object code, such as applications, are made to run through the use of an executable file. *See* Ex-1009 at 814 (“For (execute only) program text, the executable binary file (*i.e.*, .exe or .dll file) ...”); *see also id.* at 267; Ex-1010 at 44 (“Each executable image (.EXE)...”).

3. Claim 6: “A method as defined in claim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.”

198. As I previously discussed, claim 2 is obvious. In my opinion, claim 6 is also obvious.

199. **First**, as I previously discussed, Schaefer teaches that its OS Guard includes a component called the “loader subsystem 116” which “is used to allow virtual environments to be transferred into and out of the running system.” Ex-1008, [0024]. In my opinion, a POSITA would have found it obvious that in order to load and unload virtual environments, the containers of application software would need a form of unique identification. Otherwise, without having a unique identity for each virtual environment, the OS Guard (*i.e.*, its loader subsystem) would not be able to

determine which virtual environment among the many possible virtual OS environments to load. Nor would any given computer with Schaefer's networked environment be able to identify where any given virtual environment is stored/located.

200. Further, consistent with Schaefer's disclosure, Gaines also teaches that a unique identity is assigned to particular copies of the virtual OS, which in turn, uniquely identifies the virtual OS environment (*i.e.*, container). *See* Ex-1007, 11:17-22 ("Each copy of the virtual operating system [*i.e.*, virtual OS environment] 141 has a unique identifier 209."). Thus, in my opinion, a POSITA would have found it obvious that a unique identity would be assigned to each of the private contexts in Schaefer's system, such that the OS Guard can identify which private contexts to load and where they may be stored/located.

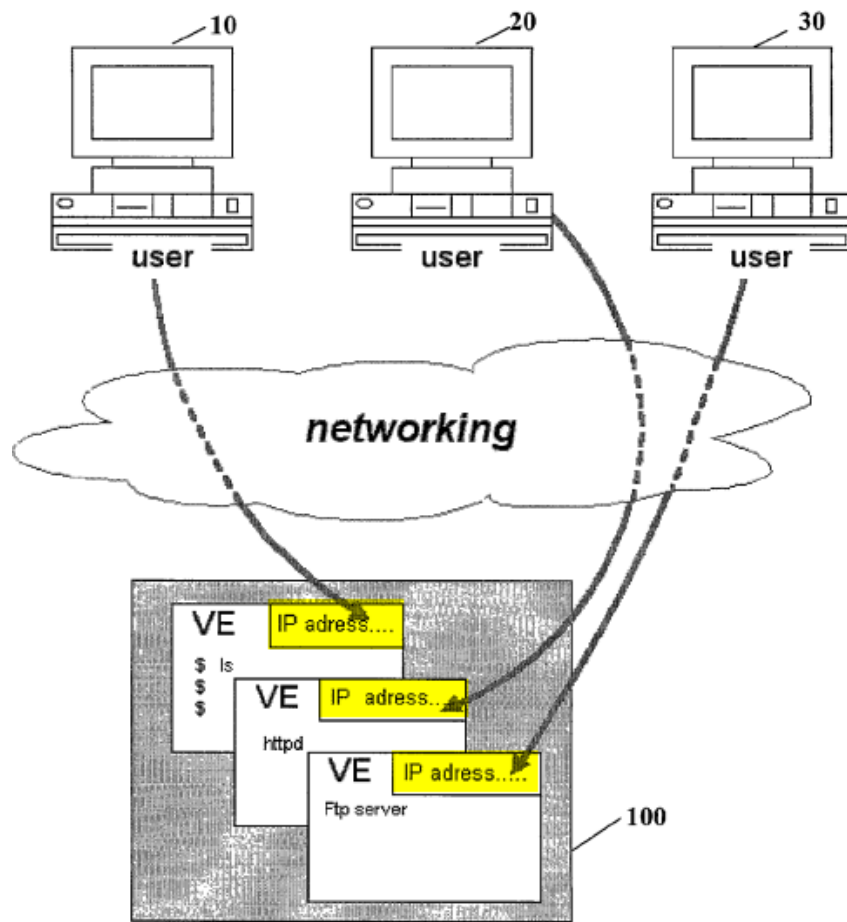
201. **Second**, in my opinion, a POSITA would have found it obvious to include in the unique identity a hostname, IP address, and/or MAC address, each of which were common prior art methods to identify resources on a computer and a computer network and pre-date the '814 Patent as discussed below. Specifically, Schaefer discloses a networked implementation. Ex-1008, [0042]; *see also id.*, [0003] (describing in the "Background of t[h]e Invention" how the claimed invention resolves issues in the prior art with respect to "environments where an application is delivered via a network"). One popular (and well-known) type of

network at the time of Schaefer was the IP network, which uses IP addresses to identify devices and other resources on the network. Ex-1009 at 556.

202. In a network configuration, a POSITA would have found it obvious that the unique identity needs to include some way for a computer seeking access to any given container to locate and communicate with the central server on which is this container stored. For example, a POSITA would have found it obvious to incorporate a network, such as one taught by Gaines, that connects Schaefer's multiple computers (*i.e.*, servers) to a central location. Ex-1007, Fig. 3A (depicting several host servers connected to a network). In any such network, the machines on the network would have addresses that identify each respective individual machine, since peer-to-peer network communication requires that each peer to be able to know which peer it is communicating with. Thus, any container on the central server (which is itself a peer within the network) would have an associated network address. One popular (and well-known) type of network at the time of Schaefer was the IP network, which uses IP addresses to identify devices and other resources on the network. Ex-1009 at 556. It would have been obvious to a POSITA to use this well-known network in Schaefer, especially given the prevalence of TCP/IP stack protocols at the time of Schaefer (which use IP addresses), so that Schaefer's uniquely identifiable virtual environments (*i.e.*, containers) could be located on the network (*i.e.*, on the central server connected to the network) and retrieved by a

computer looking to use the virtual environment (*i.e.*, container). Ex-1009 at 556 (“Since most distributed systems use the Internet as a base, the key protocols these systems use are the two major Internet protocols: IP and TCP.”), (“These are called IP addresses and form the basis of routing in the Internet.”). Gaines also expressly references use of protocols (Telnet and FTP) that use IP addresses. Ex-1007, 6:8–21

203. For the reasons articulated above, a POSITA would have found it obvious to incorporate Tormasov’s express teachings of using IP addresses to uniquely identify each of its virtual environment for use by a plurality of user computers over a network, as indicated by Figure 1 below, with Schaefer’s system.



Ex-1009, Fig. 1 (annotated), [0024] (“each virtual computing environment is the actual remote computer with the network address in which the end user can perform all actions....”). First, a POSITA would have found it obvious to incorporate a network using IP addresses, such as one taught by Tormasov, that connects Schaefer’s computers (*i.e.*, servers) to a central location, such as the hardware node taught by Tormasov, because a computer seeking access to any given container would need to be able to communicate with the central server on which this container stored. Second, a POSITA would have found it obvious to incorporate the use of IP

addresses as an identification mechanism for virtual environments, as taught by Tormasov, so that Schaefer's computers (*i.e.*, servers) are able to communicate with the central server and locate and access any particular virtual environment..

204. A POSITA would have understood that applying Gaines' teaching of a network, and Tormasov's teaching of unique identities including IP addresses, to Schaefer would be a mere combination of prior art elements according to known methods to yield predictable results (*i.e.*, unique identification of Schaefer's containers to facilitate improved access to the containers on the central server by multiple other servers on any given network using IP addresses). Combining Tormasov's teachings with Schaefer's system would also amount to the use of a known technique (*i.e.*, a networked environment and unique identities in the form of IP addresses) to improve similar devices (*i.e.*, systems comprising virtual environments that run on multiple computers) in the same way (*i.e.*, facilitating identification of and access to virtual environments stored on a central server by multiple other servers on a network). Therefore, Schaefer in combination with Gaines and/or Tormasov renders obvious assigning each of Schaefer's containers a unique identity comprising an IP address.

205. Further, in addition to IP addresses, in my opinion, a POSITA would have understood that MAC addresses were well-known address types used to identify particular devices when connected within a local network and to facilitate

communication amongst devices on the network. MAC addresses are used in many networking technologies, including Ethernet, Wi-Fi, and Bluetooth, in order to uniquely identify each device on a network and facilitate communication amongst these devices. Ex-1019, 1:6:2:11 (“Maintaining unique MAC and IP addresses for each node is critical for proper functioning of the network.”). A POSITA would have also understand that hostnames were common through the use of standalone computers and included in computers using Windows OSs, as I previously discussed.

206. Therefore, for the same reasons discussed above with respect to the use of IP addresses to uniquely identify each virtual environment (*i.e.*, container), and as I previously discussed, a POSITA would have found it obvious and beneficial to associate a hostname, IP address, and/or MAC address for each of Schaefer’s containers as part of the unique identities of the containers stored on the central server (*i.e.*, because the use of such identification mechanisms allows any of the multiple servers on a given network to locate and access the containers). In fact, IP packets typically contain both an IP address and MAC address for the destination device. Ex-1019, 1:59-2:11. Thus, a POSITA would have found it obvious to include both an IP and MAC address as part of the unique identity for each virtual environment (*i.e.*, container).

4. Claim 9: “A method as defined in claim 2, wherein server information related to hardware resource usage including

at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.”

207. As I previously discussed, claim 2 is obvious. In my opinion, claim 9 is also obvious.

208. Schaefer teaches that the system “continually monitors the use of shared system resources and acts as a service to apply and remove changes to system components.” Ex-1008, [0004]; *id.*, Cl. 4. In my opinion, a POSITA would understand that “shared system resources” would include hardware resources such as CPU memory, network bandwidth, and disk allocation, and that this information would be associated with applications running in a computer system, including in a virtual environment on the computer system.

209. Further, as discussed above, each of Schaefer’s private contexts (*i.e.*, containers) include a virtual file system. A POSITA would understand that the virtual file system would include disk allocation information. While each virtual environment believes it has its own physical memory (*i.e.*, disk storage), because the virtual environments run on a single base OS of a given host server on Schaefer’s network, the virtual environment uses the same physical memory as any other virtual OS environment running on that host server, and as such, each virtual environment requires an allocation of disk storage. Even more so, Schaefer recognizes that its claimed “invention is particularly useful for applications delivered over a network []

it is important to understand that software applications are made of several kinds of data, where the bulk of the files a software application uses are most preferably mounted on a separate logical drive.” Ex-1008, [0042]. This confirms that Schaefer’s private contexts are allocated with a portion of physical disk storage, since software applications running within the private contexts use are stored on a logical, or virtual, drive, and logical drives are part of the physical disk storage. Information regarding disk allocation would be provided by the virtual file system.

210. Further, Gaines similarly teaches constraints on resources to be made available to given applications. *See* Ex-1007, 13:52–57 (“The owner or administrator of the server host computer 101 details the constraints on resources the server host computer 101 will make available to virtual applications 143, such as limits on processor time or processor loading, processor memory space, or file system space.”). In my opinion, a POSITA would have found it obvious to use and associate server information related to hardware resource usage, including for example to monitor hardware resources, such as CPU memory, network bandwidth, and disk allocation, with Schaefer’s containers, prior to application execution, as part of Schaefer’s teachings of delivering, loading, and running virtual environments (*i.e.*, private contexts) onto any given server on that network. Otherwise, Schaefer’s containers, and applications with any given container, could not run on a given system.

5. Claim 10: “A method as defined in claim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof.”

211. As I previously discussed, claim 2 is obvious. In my opinion, claim 10 is also obvious.

212. As previously discussed, Schaefer explains that its virtual environments “prevent[] a running application from making changes to the client computer.” Ex-1008, [0013]; *see also id.* at [0019] (“[E]ach application is able to run in a private context within the system. To the application, it has its own private view of what the system looks like and its behaviors.”). Indeed, rather than to access files from the base file system or other virtual environment, Schaefer’s private contexts provide each application running within with a separate or differing copies of system services, configuration, and data. Ex-1008, [0019], Fig. 2 (showing “system files” within each “private context”).

213. Schaefer’s OS Guard plays a critical role in securing the virtual environment from the running base OS environment and other virtual environments. Specifically, Schaefer explains that its OS Guard prevents a virtual environment from making changes to the underlying OS, and any changes are limited to the virtual environment during execution of an application therein. For example, Schaefer discloses several purposes for its OS Guard, including that “it prevents a running

application from making changes to the client computer,” it “presents an environment to a running application that appears to be an installation environment without performing an installation,” and it “prevents information that may exist on the client/users machine from interfering with or modifying the behavior of an application.” Ex-1008, [0013], [0015], Cl. 2; *see also id.*, [0038] (“In general, unless specifically configured as such, shared objects should be loaded private to ensure conflict prevention. . . . if the system crashes, it is required to recover from this crash to a clean slate, not having overwritten or modified the underlying operating system.”), [0041] (“If an application tries to create a new file or open an existing file for writing on a user’s local drive, the Operating System Guard must ensure that the file is actually created or modified in the redirected location. If the application is reloaded at a later time, this file mapping must be reloaded into the Operating System Guard virtual environment. When the request is to modify an existing file, which resides on a user’s local drive, the Operating System Guard must copy the file in question to the redirection point before continuing with the request. The redirected files may not be of the same name as the original file to ensure safe mapping of file paths. In the preferred embodiment, MI files are handled in this way to offer maximum system security while allowing maximum application compatibility.”), [0052] (“[T]he Operating System Guard can use its File subsystem to avoid placing the actual font file in the running system.”). Therefore, a POSITA would understand

that in Schaefer, applications running in private contexts cannot access or modify the system files within the base OS file system, as Schaefer's OS Guard serves as a barrier between private contexts and the base OS.

214. Further Schaefer explains that although "by implementing the present invention, two instances of the application may [] run on the same client computer," "each connect[s] to a different database," and the OS Guard "keeps the two instances of the application 50 from interfering with one another." *See* Ex-1008, [0017], [0063], Figs. 1-2. Schaefer also explains that "many operating systems 10 provide fault domains to protect applications 50 from affecting each other when run . . . [a]n operating system abstraction and protection layer 100 [*i.e.*, Schaefer's OS Guard as depicted in Figure 1], will provide an additional, programmatically controlled barrier between applications 50 to remove most application level interactions." Ex. 1008, [0012], Fig. 1 (depicting an OS Guard between private contexts and between private contexts and the operating system). Therefore, in Schaefer, applications running in one private context is prevented from accessing, interfering, and/or interacting with applications and system files in another private context.

215. In my opinion, in view of the foregoing, a POSITA would understand that the applications running within a private context of a virtual environment in the combined Gaines/Schaefer system have no access to system files or applications in other containers or the underlying base OS system files during execution, because

they are part of the client computer, or alternatively, because each of Schaefer's private contexts and the contents stored therein are insulated from each other and from the base operating system in order to mitigate the possibility of conflicts and/or interference. Further, a POSITA would understand that if applications in private contexts were able to freely access other private contexts for system files, there would be no need for Schaefer to teach providing applications in private contexts with differing copies of system services, configuration, and data.

X. SECONDARY CONSIDERATIONS OF NON-OBVIOUSNESS

216. I have been informed that Patent Owner asserts that two secondary considerations of non-obviousness support a finding of non-obviousness of the '814 Patent: commercial success and a long-felt need. Ex-1015, 11-15. Notwithstanding the above discussion, which demonstrates the absence of any long-felt need for the claimed inventions because the patented features were already well-known and understood, Patent Owner does not offer substantial analysis of these secondary considerations, offering only conclusory language without showing any nexus between the claimed commercial success and long-felt need and the alleged invention of the '814 Patent. To the extent PO attempts to meet its burden of showing secondary considerations of non-obviousness, I reserve the right to respond.

Declaration of Dr. Mowry for *Inter Partes* Review of U.S. Patent No. 7,519,814

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

Dated: February 7, 2025

Respectfully submitted,



Todd C. Mowry, Ph.D.