IN THE UNITED STATES DISTRICT COURT

FOR THE DISTRICT OF COLUMBIA

STATE OF NEW YORK *ex. rel.*
Attorney General ELIOT SPITZER, *et al.*,

Plaintiffs,

v.

Civil Action No. 98-1233 (CKK)

MICROSOFT CORPORATION,

Defendant.

Direct Testimony of
John K. Bennett, Ph.D., P.E.

Table of Contents

I. Introduction

1. My name is John K. Bennett. I am a professor in the Departments of Computer Science and Electrical and Computer Engineering at the University of Colorado at Boulder. I am a registered professional engineer in the State of Texas.

2. I have been retained by counsel for Microsoft Corporation ("Microsoft") to evaluate the technical implications of certain elements of Plaintiffs' First Amended Proposed Final Judgment (the "States' Remedy") and Microsoft Corporation's Second Revised Proposed Final Judgment ("Microsoft's Remedy")[1] in the case before this Court. I have also been asked to comment on certain statements addressing technical issues that have been made by the States' witnesses in their testimony before this Court.

II. Background and Qualifications

A. Professional Background[2]

3. I began programming computers in 1969 as an undergraduate at Rice University, where I obtained a Bachelor of Science in Electrical and Computer Engineering and a Masters of Electrical

Engineering. My formal training at Rice concentrated in the area of computer system hardware and software design.

4. While I was an undergraduate at Rice University, I obtained part-time (20 hours/week during the school year, full time during the summer) employment with Texas Scientific Corporation, first as an engineering assistant, and later as a designer and programmer. During my senior year, I assumed responsibility for all prototype testing and evaluation of the company's products. Examples of projects on which I worked included the mission control system for Jet Propulsion Laboratories and the control system for the Washington, D.C. Metropolitan Transit Authority light rail system.

5. While I was obtaining my Masters of Electrical Engineering degree at Rice, I was also employed as a research and teaching assistant. During that time, I participated in the design, construction, programming and checkout of the R2 Rice Research Computer, a well-known project that pioneered what are called "tagged architectures".

6. After obtaining my Masters of Electrical Engineering degree from Rice, I was commissioned as an officer in the U.S. Navy. My initial assignment in the Navy was as the Electrical Officer of a new gas-turbine-powered destroyer. In this capacity, I managed the testing and support of the ship's computerized propulsion control system. This assignment involved the identification, documentation and correction of hundreds of design deficiencies in this new system. I later became a designated Engineering Duty Officer, and was assigned to Puget Sound Naval Shipyard. At PSNS, I served as the Senior Ship Superintendent responsible for all aspects of the repair and overhaul of two nuclear cruisers and a conventional destroyer.

7. After completing my service in the Navy, I entered graduate school in the Department of Computer Science at the University of Washington where I received my second Masters degree and my Doctoral degree, both in Computer Science. While at the University of Washington, I took advanced coursework in computer system and operating system design. I was instrumental in the design and development of the Eden Distributed Computer System, one of the first object-oriented distributed computing systems ever developed. My doctoral dissertation involved the design, implementation and evaluation of another object-oriented distributed computing system based upon Smalltalk. Smalltalk is an object-oriented programming environment developed at the Xerox Palo Alto Research Center that was a precursor of Sun's Java technology.

8. Prior to completing my Doctorate at the University of Washington, I founded and served as President of Pacific Mountain Research, Inc. ("PMR") in Seattle, Washington. At PMR, I directed the design and development of hardware and software components of a variety of commercial computer systems including: (1) multi-processor and virtual memory systems, (2) several high performance graphics and image processing systems, (3) custom very large scale integrated (VLSI) circuits, (4) biomedical instrumentation and (5) video processing systems. PMR designs were utilized or marketed by several companies, including Motorola, Signetics, RCA, IBM, Eastman Kodak, Microsoft, Sequent, Weyerhaeuser, the Department of Defense and the New York Stock Exchange. PMR was frequently responsible for the testing and initial software and hardware support of these systems.

9. After completing my Doctorate, I joined the faculty of the Department of Electrical and Computer Engineering at Rice University in 1988. At Rice, my research focused broadly in the area of operating system/microprocessor architecture interaction and more narrowly in the area of distributed and parallel computing systems. With my students, I developed the Munin distributed shared memory ("DSM") system, the prototype for most modern software DSM systems, and the Brazos Parallel Programming Environment, which supports clusters of multiprocessor computers (multicomputers) running Windows NT and Windows 2000. Brazos is in use by a number of academic and research users around the world. My students and I also developed several mechanisms that exploit high-performance user level networks. One aspect of this work is the implementation of a better-performing alternative to Microsoft's Windows Sockets Direct Path, a high-performance networking protocol. Other work I did at Rice concentrated on building efficient and reliable execution platforms for parallel applications on clusters of multicomputers, fault-tolerant parallel robotics and detecting holes in surgical gloves during use. During my last four years at Rice I served as the Master of Richardson College.

10. Two years ago I joined the faculty in the Departments of Computer Science and Electrical and Computer Engineering at the University of Colorado at Boulder, where I teach, among other subjects, the graduate Advanced Operating Systems class. My current research focuses broadly in the area of distributed information management, and more narrowly in two areas: (1) a project called Bifrost that involves the development of a system that facilitates ubiquitous location-independent access to user information, and (2) the development of assistive technology for persons with disabilities that employs mobile computing and wireless technologies.

11. My work has been supported over the years by the National Science Foundation, the Defense Advanced Research Project Agency, NASA, the National Institutes of Health and the Texas Higher Education Coordinating Board, as well as by grants from Compaq, Intel, Tandem, Microsoft, Ansell, Regent, Bell Northern Research, Hewlett-Packard and IBM. I have published over 35 research papers, and have given presentations at numerous conferences, universities and research laboratories.

12. As a result of this experience, I am thoroughly familiar with the cycle of hardware and software development, testing and support. I have participated in the development and testing of commercially available software products as both a software developer and manager. In the course of my research and teaching, I have also become broadly familiar with various client and server operating systems, and a variety of programming environments, including Microsoft's new .NET initiative.

B. Cases in Which I Have Testified at Trial or by Deposition as an Expert

13. I have testified as an expert in seven other cases. These are:

Haden and Company v. AMS (1990)

ResTech v. *Joseph Hawkins* (1993)

Siege Industries, Inc. v. *Clark Manufacturing* (1996)

Caldera, Inc. v. *Microsoft Corporation* (1997)

Bristol Technology, Inc. v. *Microsoft Corporation* (1999)

Compaq Computer Corporation v. *eMachines, Inc.* (2000)

Compaq Computer Corporation v. *Unova, Inc. et al.* (2001)

14. In these cases, my testimony related to issues of software behavior and compatibility, and to intellectual property concerns. I have provided consulting and expert opinion not involving testimony to several other companies, however this other work is governed by protective orders or non-disclosure agreements.

III. Materials Considered

15. My knowledge in these areas (and, therefore, my opinions) derives in part from my active involvement in computer systems research and development for over twenty years. For fourteen of those years I have taught classes addressing computer system design and development issues (including those relating to operating systems). As part of my teaching and academic advancement, I keep current on issues in computer software, including (specifically) operating system development.

16. Aside from my background in computer science and engineering, in the course of preparing this testimony I have taken into consideration the following materials:

a. Plaintiffs' First Amended Proposed Final Judgment ("States' Remedy");

b. The Second Revised Proposed Final Judgment ("Microsoft's Remedy") between Microsoft Corporation, the United States of America and the States of New York, Ohio, Illinois, Kentucky, Louisiana, Maryland, Michigan, North Carolina and Wisconsin;

c. The rulings of the United States District Court for the District of Columbia in *United States* v. *Microsoft Corp.*, including the Findings of Fact, 84 F. Supp. 2d 9, Conclusions of Law, 87 F. Supp. 2d 30, and June 7, 2000 Memorandum and Order, 97 F. Supp. 2d 59;

d. The June 28, 2001 decision of the United States Court of Appeals for the District of Columbia Circuit in *United States* v. *Microsoft Corp.*, 253 F.3d 34;

e. The written direct testimony and transcript of the trial testimony of Edward W. Felten in the liability phase of this case;

f. The Supplemental Response of the Plaintiff Litigating States to Defendant Microsoft Corporation's First Set of Written Interrogatories and Responses of Plaintiff Litigating States to Defendant Microsoft Corporation's Second Set of Written Interrogatories;

g. Microsoft Corporation's Response to Plaintiff Litigating States' First Set of Interrogatories in Remedy Proceedings, served January 11, 2002;

h. The source code for Windows XP Professional and what is referred to as the source code for Windows XP Embedded (which is actually the source code for the tool suite);

i. The deposition testimony and the ongoing trial testimony of a number of witnesses in this case, particularly those specifically cited in my testimony;

j. Publicly available documents relating to issues in the case, including press articles, books and web sites;

k. Other documents and information specifically cited in my testimony;

l. Telephone interviews that I conducted of various Microsoft personnel, which are cited in context in my testimony; and

m. Certain other bases for my opinions in the context of the particular testimony that I offer below.

IV. Summary of Testimony

17. Based upon my examination and consideration of the documents and materials detailed above, and upon my professional experience and training, my testimony in this matter can be summarized as follows:

a. Section 1 of the States' Remedy is Technically Infeasible.

b. The States' Remedy Definition of "Bind" and "Microsoft Middleware Product" Render Section 1 of the States' Remedy Unworkable.

c. Section 1 of the States' Remedy Would Negatively Impact Software Developers and Consumers.

d. Section 1 of the States' Remedy Would Severely Impact the Ability to Test Windows Operating System Products, with a Resulting Direct Negative Impact on Consumers.

e. Section 1 of the States' Remedy Would Create an Unreasonable Support Burden, Which Would Also Negatively Impact Consumers and OEMs.

f. Windows XP Embedded Does Not Demonstrate that Section 1 of the States' Remedy is Technically Feasible.

g. The Experiments Conducted by Dr. Edward Felten Do Not Establish that Section 1 is Technically Feasible.

h. Dr. Andrew Appel's Suggested Methods for Microsoft to Comply with Section 1 of the States' Remedy Fail Upon Close Analysis.

i. Section 2 of the States' Remedy is Technically Infeasible.

j. Section 2 Exacerbates the Support Problems Created by Section 1 of the States' Remedy.

k. The Definition of "End User Access" Renders Section 2 Infeasible.

l. Section 3 of the States' Remedy Creates an Unreasonable Support Burden.

m. Section 4 of the States' Remedy is Technically Unreasonable.

n. There are Sound Technical Reasons for Operating Systems Vendors to Decline to Make Internal Interfaces Public.

o. The States' Remedy Definition of "API" is Overly Broad.

p. The States' Remedy Definition of "Communications Interface" is Both Overly Broad and Technically Flawed.

q. The States' Remedy Definition of "ICP" is Overly Broad.

r. The States' Remedy Definition of "IHV" is Overly Broad.

s. The States' Remedy Definition of "Interoperate" is Overly Broad.

t. The States' Remedy Definition of "ISV" is Overly Broad.

u. The States' Remedy Definition of "OEM" is Overly Broad.

v. The States' Remedy Definition of "Technical Information" is Overly Broad.

w. The States' Remedy Definition of "Timely Manner" Would Stifle Microsoft's Ability to Innovate in Windows Operating Systems.

x. The States' Remedy Definition of "Microsoft Platform Software" is Overly Broad.

y. The States' Remedy Definition of "Operating System" is Overly Broad.

z. The States' Remedy Definition of "Platform Software" is Overly Broad.

aa. The States' Remedy Definition of "Web-Based Software" is Overly Broad.

bb. Section 4 of the States' Remedy is Technically Infeasible.

cc. Section 5 of the States' Remedy is Technically Infeasible.

dd. The Scope of Section 5 of the States' Remedy is Unreasonably Broad.

ee. Section 4 of the States' Remedy Exacerbates the Negative Effect of Section 5.

ff. Section 10 of the States' Remedy is Technically Infeasible.

gg. Section 12 of the States' Remedy is Technically Problematic.

hh. Section 13 of the States' Remedy is Technically Flawed.

ii. Section 5 of the States' Remedy is Technically Flawed.

jj. Office for Macintosh is Not a Port of Office for Windows.

kk. Section 14 of the States' Remedy Does Not Recognize Important Differences Between the Windows and Macintosh Office Products.

ll. Section 14 of the States' Remedy Would Likely Require Significant Disclosure of Windows Operating System Source Code.

mm. A number of the technical issues related to the behavior of Microsoft software raised in the testimony of Plaintiffs' witnesses in this case have either mischaracterized the relevant behavior of the Microsoft software at issue, or have not included important and relevant related information.

18. I will explain the rationale for each of these opinions in the remainder of my testimony.

V. Technical Feasibility of the States' Remedy and Microsoft's Remedy

A. Software Development Issues

19. Writing software is a complex and labor-intensive enterprise. When creating new software (or when modifying existing software), the software developer must take into account various constraints:

· The target hardware architecture, including both the computer itself, as well as any associated peripheral devices;

· The available software development platforms and the functionality exposed by those platforms to other software;

· The intended use and intended user of the software under development;

· Whether or not the software is intended to provide functionality to other software by exposing an application program interface ("API"), and whether the other software is "trusted" (known to behave in a reliable manner) or "untrusted" (every attempt by the other software to access the functionality provided by the API must be verified for correct behavior);

· Whether or not the software will use functionality provided by APIs other than those exposed by the relevant software development platform;

· The mechanisms by which the software under development will communicate with other software, both that installed on the same computer and that installed on other computers;

· Whether or not the software must be backwardly compatible with other software that may be used on the computer;

· Whether or not the software must be protected from unauthorized or unsafe access, and whether or not the software will employ security mechanisms in order to gain access to other software; and

· Non-technical constraints including time to market and budget limitations such as those imposed by customer demand or competing software.

20. Virtually all of these constraints come into play when the software in question is an operating system or a component of an operating system. Further, because an operating system by design provides services to other software, the details of which are generally not known, application software developers must interact with the operating system in well-defined ways. For this purpose, operating system developers typically publish APIs that contain a set of system services intended for use by application programs. When an application program makes a call to one of these system services, the operating system performs a number of checks to ensure that the call is valid. These checks typically include:

· Whether the caller is authorized to make the indicated call;

· Whether all of the parameters passed with the call are valid (e.g., does a parameter that is supposed to point to memory in fact point to a valid region of memory?);

· Whether the parameters are all of the correct type; and

· Whether it is safe to make the changes in system state that executing the call will involve.

21. These checks are required in order to ensure that the operating system will continue to function correctly, but necessarily take time to perform. If the behavior of calling software is known completely - for example, if the calling software is a trusted component of the operating system - then it is possible to "short-circuit" some of these checks in order to improve the performance of the operating system. I stress that this is only possible if it is absolutely certain that the calling software will behave in a known way, thereby rendering the checks unnecessary. Recognizing this distinction, operating system designers frequently create two similar versions of

the same system call, one in which the checks are performed, and one in which the checks are not performed. The version in which the checks are performed is the only one that is safe to make "public"; to do otherwise could potentially allow an application program to render the operating system unusable. The "private" interface is reserved for the exclusive use of trusted operating system components.

22. I provide this background because it is central to an understanding of certain issues in this case.

B. Section 1 of the States' Remedy is Technically Infeasible.

1. The States' Remedy Definition of "Bind" and "Microsoft Middleware Product" Render Section 1 of the States' Remedy Unworkable.

23. By virtue of the definition of "Windows Operating System Product" in Paragraph 22.rr, Section 1 of the States' Remedy would require Microsoft to create and support "both directly and indirectly", an "unbound" version of Windows 95,[3] Windows Millennium Edition, Windows 2000 Professional, Windows XP Home, Windows XP Professional and their successors, such that "the binary code for each Microsoft Middleware Product (including any code providing similar functionality that has been included in any other Microsoft Middleware Product) may be readily removed, such that this 'unbound' Windows Operating System Product performs effectively and without degradation (other than the elimination of the functionalities of any removed Microsoft Middleware Products)." After a six-month delay, this requirement would be imposed whenever Microsoft chose to "Bind" "any Microsoft Middleware Product to the Windows Operating System." In addition, after the same six month delay, Microsoft would be required to "make available a Windows Operating System Product that permits the removal of"[4] "Internet browsers, e-mail client software, media creation, delivery and playback software, instant messaging software, voice recognition software, digital imaging software, directories, Exchange, calendaring systems, systems and enterprise management software, Office, Handheld Computing Device synchronization software, directory services and management software, the Common Language Runtime component of the .Net framework, and Compact Framework."[5]

24. Putting aside significant issues of ambiguity, the technical infeasibility of Section 1 of the States' Remedy derives in large part from its extraordinarily broad definitions of "Bind" and "Microsoft Middleware Product". I will address each of these definitions in turn.

25. The States' Remedy essentially defines "Bind" as including software in an operating system that cannot be removed without degrading the performance or impairing the functionality of the operating system. In their amended remedy, the non-settling States appear to have at least recognized that it is not possible to remove software while at the same time expecting that software to continue to function, but they continue to ignore the significance of the fact that removing software from an operating system is likely to cause other software to cease to function. This is because such other software may depend upon certain functionality provided by the software that is being removed. This dependence is based upon the sound software engineering principles of code sharing and code reuse. It is, however, generally feasible to remove or disable the end-user-visible portions of software functionality without rendering other dependent software

inoperative. That is the approach taken by Section III.H of Microsoft's Remedy, which allows either end users or OEMs to remove direct end-user access to "Microsoft Middleware Products". There are significant differences between these approaches, for reasons I explain more fully below.

26. A simple example will illustrate some of these issues. Consider Acme Software Company, which sells a software calculator with five functions: addition, subtraction, multiplication, division and square root. Now suppose that Acme is required to remove all of the code related to the square root function. In designing the software calculator, however, Acme employed a common design methodology known as "code reuse". All of the five calculator functions share certain underlying code, and there is no differentiation in this underlying code between any of the five operations. Such sharing is good practice from a software engineering perspective, for two reasons: it lessens the presence of redundant code, thereby shrinking the "memory footprint" of the calculator software; and it makes the calculator software easier to maintain, because when it is time to update or modify the calculator software, the functionality of interest will be found in just one location in the code, rather than being replicated in many places in the code. In fact, the potential divergence of replicated code over time is a well-known software engineering problem that software developers strive to avoid.

27. The "removal" of all of the code associated with the square root function (including the underlying code) would necessarily degrade, or even render inoperative, the other four functions, because of code sharing and reuse. On the other hand, it would be relatively straight-forward to remove direct access to the user-visible portion of the square root function (e.g., by deleting the square root button from the calculator's on-screen display), making it appear to the user that the square root function has been removed, but without affecting the other functions. Removing the user-visible part of the square root function is also a more easily defined undertaking for Acme, since this aspect of the square root functionality is readily identifiable as unique to the square root function. Disabling the square root function in this way also makes it more likely that a user will employ software from one of Acme's competitors to compute square roots, because the user will be unaware that Acme's software can perform this function. Using two products rather than one, however, may still be sub-optimal, as it will potentially deny to the user the various benefits of software integration, including the code sharing noted above and the consistency of operation that is likely to result from having all five functions performed by the same software. Nevertheless, this alternative remains superior to the removal of software code envisioned by Section 1 of the States' Remedy.

28. The practical differences between removing code as in the States' Remedy, and removing or disabling end user access as in Microsoft's Remedy, is further illustrated in Figure 1, below. I will discuss this Figure briefly in oral direct testimony. This Figure shows in a very simplified manner why removing Internet Explorer code from Windows XP will

degrade or break operating systems functions and applications that rely upon APIs supported by that code. Figure 1 compares this situation to the impact of removing or disabling end-user access

to the Internet Explorer user interface, in which case some other browser, such as Netscape Navigator, can be used to provide a browsing user interface, while preserving the functionality of other operating system functions and the user's application programs.

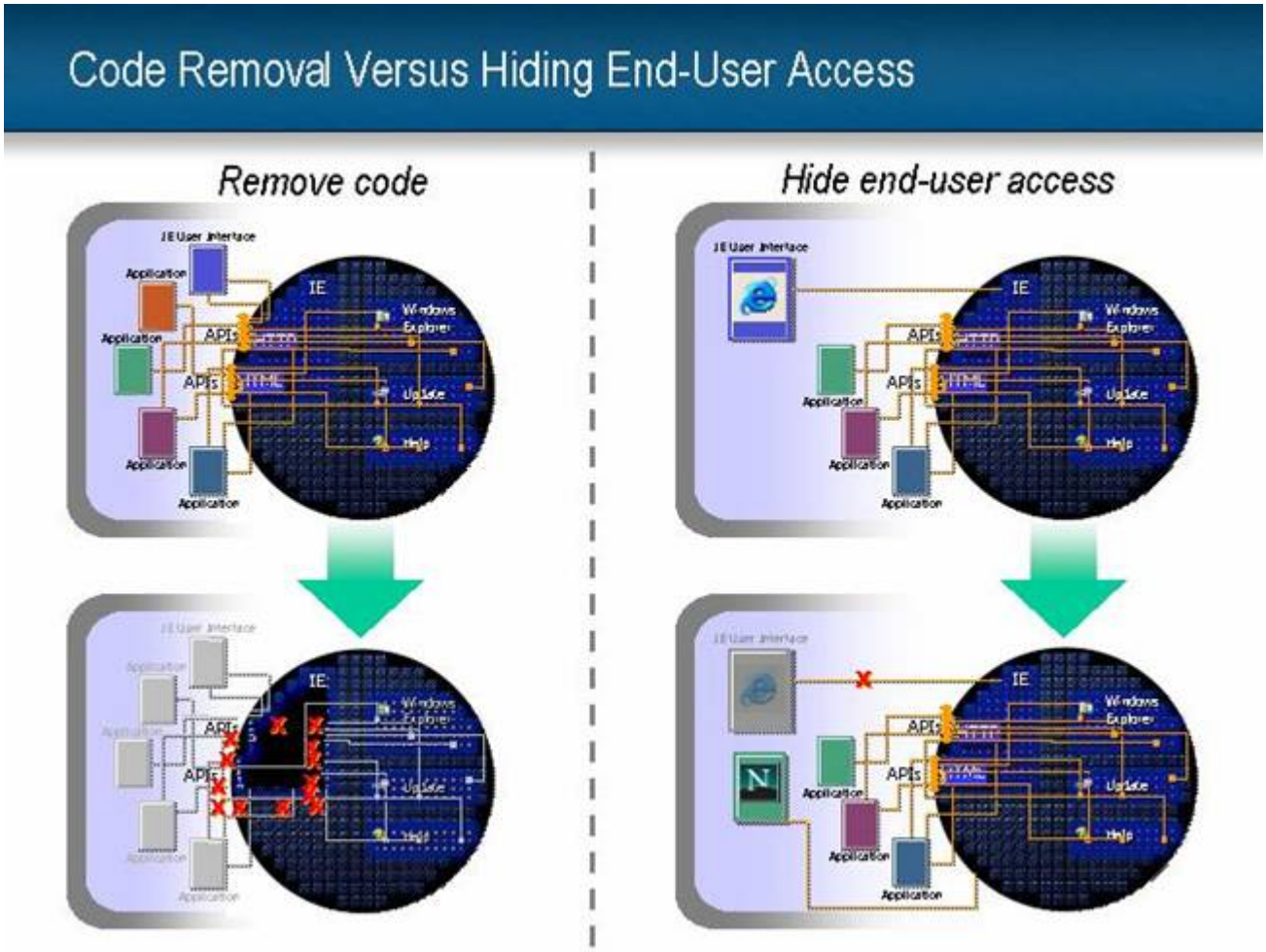

Figure 1

29. The States' Remedy definition of "Microsoft Middleware Product" is multifaceted. Paragraph 22.x(i) includes "Internet browsers, e-mail client software, media creation, delivery and playback software, instant messaging software, voice recognition software, digital imaging software, directories, Exchange, calendaring systems, systems and enterprise management software, Office, Handheld Computing Device synchronization software, directory services and management software, the Common Language Runtime component of the .Net framework, and Compact Framework, whether provided in the form of files installed on a computer or in the form of WebBased Software." Paragraph 22.x(ii) includes "Middleware" (an even more broadly defined term) distributed by Microsoft that "(1) is, or in the three years preceding this Judgment has been, distributed separately from an Operating System Product, any successors thereto, or (2) provides functionality similar to that provided by Middleware offered by a Microsoft competitor."

30. The definitions of "Middleware" and "Microsoft Middleware Product" lie at the heart of many of the technical flaws in the States' Remedy. The definition of "Middleware" appears to

include within its scope all of the code in Windows, and all other Microsoft products, that exposes external functionality that can be invoked by other software. For example, since "Middleware" includes software "that operates directly or through other software within an Operating System"[6] and that "could, if ported to or made Interoperable with multiple Operating Systems, enable software products written for that Middleware to be run on multiple Operating System Products", substantial portions of the code that supports the Win32 API set (the core functionality used by all Windows application developers), as well as portions of the Windows operating systems kernel, would be "Middleware" under the States' Remedy.

31. Figure 2 illustrates this problem. In Figure 2, I have depicted just a few of the key software modules in Windows operating systems that expose APIs to application programs and other modules of the operating system. Arrows on the diagram depict call dependencies, i.e., if an arrow is drawn from user32.dll to kernel32.dll, functions in user32.dll call functions in kernel32.dll. Five of the modules shown execute in user mode (user32.dll, advapi32.dll, gdi32.dll, kernel32.dll and ntdll.dll), and three modules execute in kernel mode (ntoskrnl.exe, win32k.sys and hall.dll). The kernel-mode module called hall.dll provides an abstraction of the underlying hardware to the rest of the operating system.

32. All of the user-mode modules expose APIs that can be invoked by application programs. Most application programs requiring code to execute in kernel mode ultimately invoke code in ntdll.dll that in turn invokes code in ntoskrnl.exe through a special software interrupt, but the graphics subsystem is an exception to this rule. The user-mode part of the GDI (graphics display system) in gdi32.dll makes direct calls to the kernel-mode part of the GDI (found in win32k.sys). The kernel-mode part of the GDI is also called directly by code in user32.dll. User-level communications software behaves in a similar manner.

33. Each of the eight modules shown in Figure 2 meets the definition of "Middleware" contained in the States' Remedy, since, if ported, it would "enable software products written for that Middleware to be run on multiple Operating System Products." Further, the States' Remedy also defines these modules as "Microsoft Middleware Products", and thus would require them to be optionally removable under Sections 1 and 2.c of the States' Remedy, since they are "Middleware distributed by Microsoft" that "provides functionality similar to that provided by Middleware offered by a Microsoft competitor."[7] I understand that the non-settling States have argued that if the removal of components of Windows that fall within the definition of "Microsoft Middlware Products" will cause the operating system to malfunction, then OEMs and Third-Party Licensees will not remove those components. This argument appears to miss the point. Section 1 requires Microsoft to design, develop and test "unbound" versions of several different Windows operating systems to ensure that every component defined as a "Microsoft Middleware Product" can be removed without impairing the performance or degrading the functionality of the operating system. It is that task that is technically infeasible to accomplish.

34. These modules, as well as many others not depicted in Figure 2 that would similarly fall under the definition of "Microsoft Middleware Product" in the States' Remedy, form the foundation of all of the tens of thousands of applications written for the Windows operating system. If any of the eight modules I have referred to in Figure 2 is removed from Windows, the operating system itself will not run, and applications that rely on the APIs exposed by these

modules will cease to operate, resulting in a serious negative impact on the consumers who rely on Windows applications in their daily lives.

Win32 Core API Modules (many not shown)

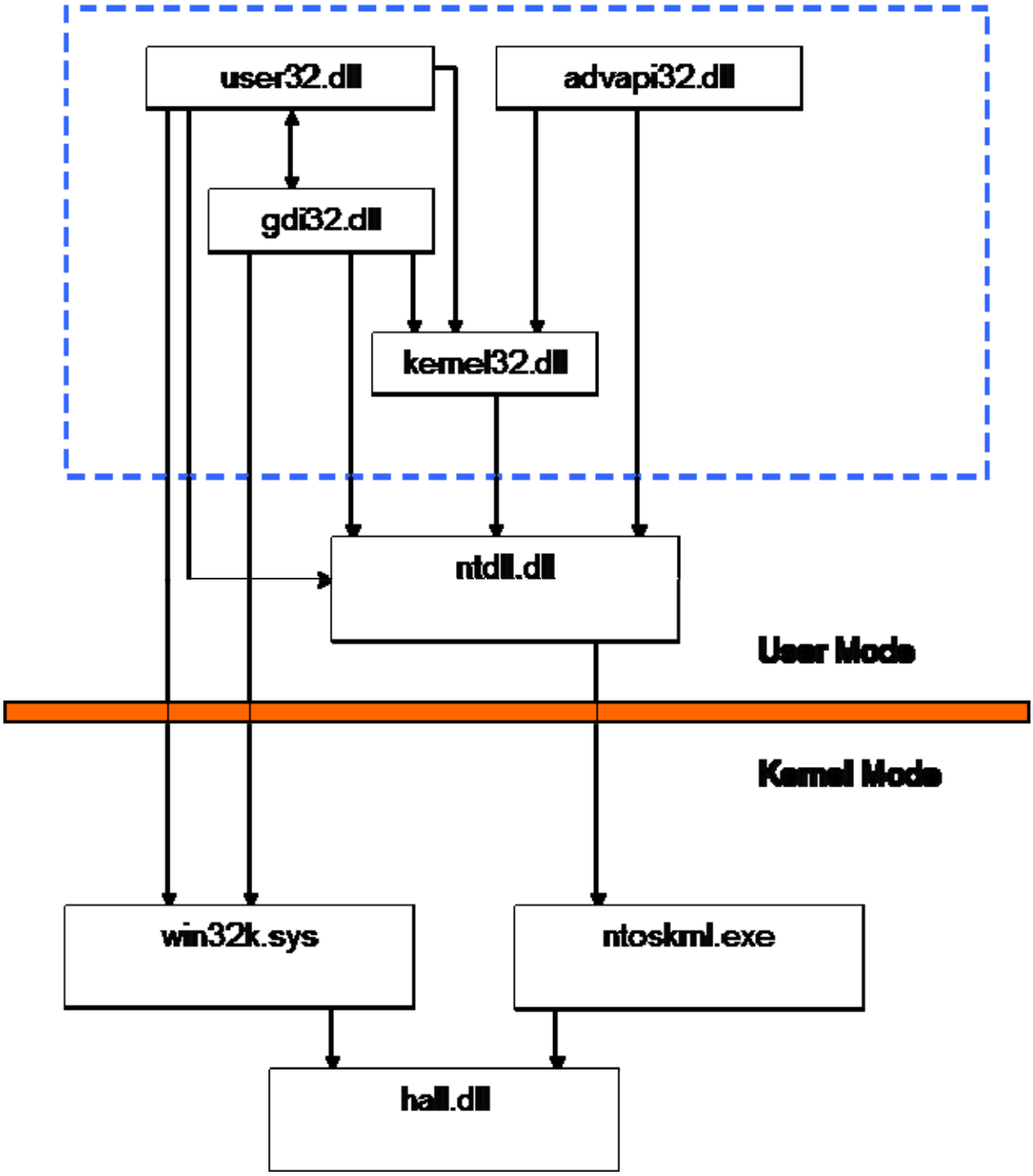

Figure 2: Windows OS Modules Defined as "Middleware" by the States' Remedy

35. The non-settling States apparently have taken the view that any piece of code that exposes an API (here again, the definition of "API" in the States' Remedy is quite broad) is "Middleware",

even though it appears they have not always held this view. In an earlier filing in this case, *Plaintiff Litigating States' Remedial Proposals*, the non-settling States provided a far more appropriate definition of "middleware products", namely, "software that can itself be a platform for applications development".[8] This is in my view a more sensible definition of middleware, namely, software that exposes a sufficiently diverse set of APIs so as to permit software designers to develop general purpose applications.

36. Like the definition of "Middleware", the definition of "Microsoft Middleware Product" found in the States' Remedy is virtually unbounded, including essentially all of Microsoft's current and future products. In addition, Clause ii(1) of Paragraph 22.x does not qualify that the separate distribution has to have been by Microsoft, nor does Clause ii(2) of Paragraph 22.x require that the product providing the similar functionality had to do so on a standalone basis. Thus the States' Remedy definition of "Microsoft Middleware Product" appears to include any functionality found in any Microsoft product that exposes APIs and for which similar functionality existed in any product of any Microsoft competitor. That is a very sweeping definition.

2. Section 1 of the States' Remedy Would Negatively Impact Software Developers and Consumers.

37. The ease of developing applications is, in general, directly proportional to the richness of the APIs provided by the underlying operating system. In developing applications, software developers rely upon these APIs to be stable, and depend on the fact that API calls performed by their code will in fact find the correct functionality when invoked. In the absence of this assurance, software developers would have to spend significant additional time and resources to implement equivalent functionality themselves in order to ensure that their applications will work as intended. This duplication of effort would also lead to many different implementations of the same functionality, creating the potentially confusing situation in which similar operations (like menus use, user help, document cutting and pasting, etc.) work differently with different applications.

38. To use a specific example that has been raised in this case, consider the Windows "Help" system. The Windows Help system provides a common framework for applications developers to provide a searchable index of information about their products. The presence of the Windows Help system directly benefits consumers because it makes software products easier to use and because the mechanisms for obtaining help are common across applications. For the reasons stated above, the Windows Help system is also of direct benefit to software developers. However, the Windows Help system is HTML-based. The ability to interpret and display HTML in Windows is provided by the same operating system code that is used to browse the web, in this case a file called MSHTML.DLL. If this file were removed in an effort to remove "web browsing functionality" from Windows, then the Windows Help system would cease to function. In the presence of this possibility, software application developers would have to provide an application-specific Help system, removing both the benefit to the consumer (commonality of Help system behavior) and the benefit to the software developer (reduced application development time and cost). In contrast, it is possible to hide direct user access to the web browsing functionality of Windows (called Internet Explorer), without removing the native ability to render HTML. This is the approach taken in Microsoft's Remedy.

3. Section 1 of the States' Remedy Would Severely Impact the Ability to Test Windows Operating System Products, with a Resulting Direct Negative Impact on Consumers.

39. Testing modern operating systems is both a complex and time-consuming task. It is difficult to anticipate all of the ways in which the operating system will be used so that these uses can be tested. Operating systems are particularly difficult to test, in part because of their inherent complexity, and in part because during testing much of their operation and behavior must be inferred from the behavior of other software. Operating system software is especially difficult to test when the hardware on which the operating system is expected to execute can vary. That hardware variability is particularly acute in the case of personal computers, because there are literally thousands of different devices like printers, modems, scanners, network cards and video display monitors in use by consumers.

40. Although I have tested many software products, including operating systems, I have no experience testing software or operating systems at the scale applicable to modern commercial operating systems with hundreds of millions of users. In order to gain a better understanding of commercial operating systems testing in this context, and of the testing of Microsoft operating systems in particular, as I and other computer scientists often do in forming opinions on technical matters, I spoke with knowledgeable personnel at Microsoft. I conducted a telephone interview of Sivaramakichenane Somasegar, the Microsoft Corporate Vice President responsible for Windows testing, and Darren Muir, Director of Windows Integration Services, on April 25, 2002. I have in part relied upon factual information provided by these individuals in the preparation of this portion of my testimony.

41. In contrast to the proprietary and stable hardware platforms on which Sun Microsystems and Apple Computer operating systems run, Windows operating systems are expected to run on a huge variety of hardware configurations that include thousands of hardware devices not manufactured by Microsoft. This diversity of hardware imposes a significant testing burden upon Microsoft. For example, Bill Gates testified that Microsoft spent approximately $500 million testing Windows XP, and that this testing required about twenty months to complete.[9] Mr. Gates also testified that Microsoft "generally put[s] more testers than developers" on the project of developing a new operating system.[10]

42. The key criteria in operating system testing are reliability, compatibility and stability. The process of testing a new Windows operating system involves a number of steps, many of which proceed in parallel. These steps include the following:

· Internal Test Suites – special purpose programs designed by Microsoft employees that test basic operating system functionality.

· Applications Compatibility Testing – testing the new operating system by running approximately two thousand third-party applications chosen based upon application market share and the degree to which the application stresses the operating system in "interesting" ways or exhibits "special needs".

· Hardware Compatibility Testing – testing the new operating system on a broad assortment (approximately ten thousand) of different hardware configurations.

· Stress Testing – a testing suite that simulates three to six months of continuous customer use of the operating system. Approximately sixteen hundred different machines are used for this purpose.

· Self Hosting – having large numbers of Microsoft employees exercise the new operating system in daily use. For example, approximately three thousand Microsoft employees used Windows XP Professional, and approximately two thousand employees used Windows XP Home, for over a year prior to the release of Windows XP to the public.

· Beta Testing – a program (or sequence of programs) of managed releases of a new operating system still under development in which beta testers provide extensive feedback to Microsoft regarding the performance of the new operating system. For example, Mr. Gates testified that approximately 500,000 copies of beta versions of Windows XP were distributed.[11] The first beta release to 150,000 or more beta testers of Windows XP occurred more than 9 months before the product was commercially released.

· Testing by Key "Partners" – the extensive testing of the new operating system by large volume OEMs and other entities with significant internal testing requirements, such as large enterprise customers.

43. New operating systems move from the externally visible milestones of first beta release to second beta release to the product's release to manufacturing based upon variable threshold criteria. For example, a new operating system might require a pass rate of 90% to move to first beta release, a pass rate of 95% to move to second beta release, and a pass rate near 100% to be released to manufacturing. Internal milestones have increasingly higher pass thresholds as the new operating system moves closer to completion.

44. Is it not possible to completely test every aspect of a new version of Windows. This is due to the large number of possible applications and hardware configurations, the underlying dependencies within the operating system, and the fact that it is impossible to predict all the ways in which the interfaces exposed by the operating system will be used. However, Microsoft appears to test as completely as it can, and has developed a variety of mechanisms to achieve that goal.

45. Given this significant testing burden as background, consider the impact that Section 1 of the States' Remedy would have upon the testing of Windows operating systems. If there are two software components that may be absent or present in the operating system, then there are four versions (2^2) of the operating system to test: (1) the version with both components present, (2) the version with both components absent, (3) the version with only the first component present and (4) the version with only the second component present. This is illustrated in Figure 3, below.

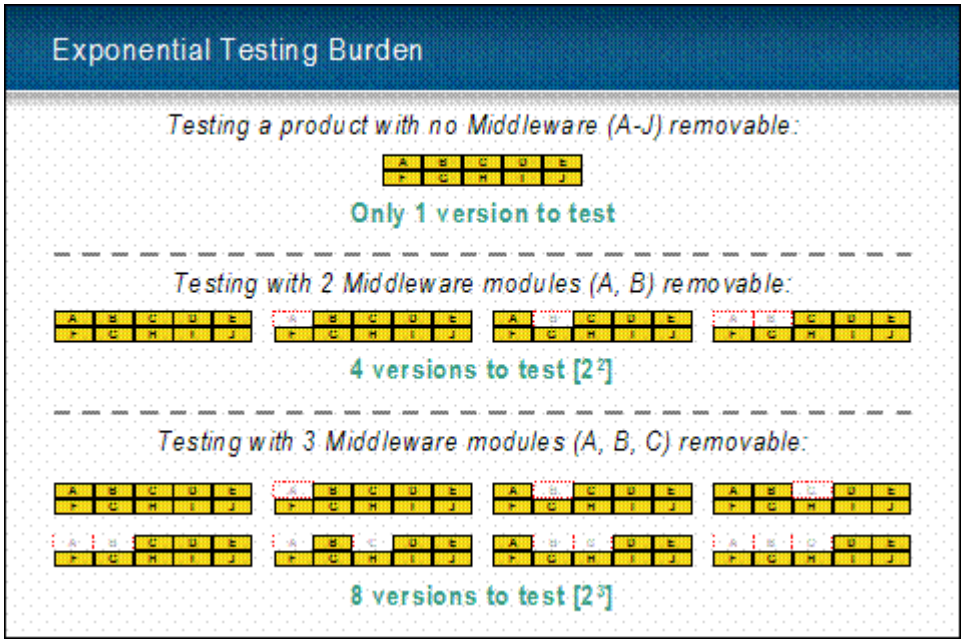

Figure 3

46. This exponential relationship continues as the number of possible components increases: for 4 components there are 16 operating system versions, for 8 components, there are 128 versions, for n removable components there are 2^n operating system versions. Now consider the number of operating systems versions that would result from an application of Section 1 of the States' Remedy.

47. Because the States' Remedy defines a "Microsoft Middleware Product" to include, among other things, "Internet browsers, e-mail client software, media creation, delivery and playback software, instant messaging software, voice recognition software, digital imaging software, directories, Exchange, calendaring systems, systems and enterprise management software, Office, Handheld Computing Device synchronization software, directory services and management software, the Common Language Runtime component of the .Net framework, and Compact Framework…", there are likely far more than ten such "Microsoft Middleware Products"[12], and ten removable components would yield 2^{10}, or 1,024 versions of Windows to test.

48. In summary, the exponential relationship between removable components and operating systems variants means that Section 1 of the States' Remedy would create an intractable testing burden for Microsoft. Microsoft Corporate Vice President Somasegar estimated that each operating system variant would add four to six months to the Windows test cycle. This figure is supported by the Windows 2000 Server product release dates: after Windows 2000 Server and Advanced Server were released at the same time, Windows 2000 Datacenter, the most advanced version of Windows 2000 server, required an additional eight months of testing before it was released.

49. Further, Dr. Andrew Appel, the non-settling States' technical expert, testified that Section 1 of the States' Remedy would obligate Microsoft to test the "different permissible configurations" "to the same extent" that it tested its existing operating systems:

"It's my belief that Microsoft's obligation is to do sufficient testing to assure that the different permissible configurations of the operating system work well to the same extent that Microsoft does sufficient testing to ensure that the different configurations of the operating systems that it already sells or that it would sell as the bound version would work well."[13]

50. Therefore, under Section 1 of the States' Remedy, Microsoft must either test much less, which has both a negative impact on consumers (and is prohibited by Section 1, since an operating system that is tested less will likely not perform "effectively and without degradation"), or will never again be able to develop and ship a new version of Windows. It is worth noting that the testing burden is not increased nearly as much if only direct user access to Microsoft Middleware is removed (as is called for in the Section III.H of Microsoft's Remedy[14]), since the operating system can be tested with all software components present, the direct user access to some of which may subsequently be disabled. In that scenario, both the operating system code and Windows as a software development platform remain stable and consistent.

4. Section 1 of the States' Remedy Would Create an Unreasonable Support Burden, Which Would Also Negatively Impact Consumers and OEMs.

51. Although I have supported many software products, including operating systems, I have no experience supporting software or operating systems at the scale applicable to modern commercial operating systems with hundreds of millions of users. In order to gain a better understanding of commercial operating systems support in this context, and of the support of Microsoft operating systems in particular, I, in accordance with my practice in developing opinions on such matters, conducted a telephone interview of Lori Moore, Microsoft Vice President of Product Support Services, and Walid Abu-Hadba, General Manager, Americas Global Technical Center, on April 29, 2002. I have in part relied upon factual information provided by these individuals in the preparation of this section of my testimony.

52. Windows operating system support is already quite complicated. The unpredictable interaction of thousands of applications and devices with the operating system creates a support problem no less challenging than the testing problem. A brief overview of Windows operating system support is helpful in understanding this issue.

53. The various Windows operating systems are supported by a staff of approximately 6,500 people at Microsoft, and another 1,500 or so people (in the U.S.) working as Microsoft-trained out-sourced support vendors.[15] The total cost of the support operation to Microsoft is about $800 million per year. Microsoft estimates that they handle only 10% of the volume of support calls generated by Windows customers. The other 90% of the call volume is handled by the OEMs who ship Windows installed on their products. Thus, the total cost of providing support for Windows is a staggering figure.

54. The call volume associated with Windows support is enormous. In addition to calls directly related to Windows, Microsoft customer support receives a significant number of calls related to third party drivers, third party applications and third party online services like AOL. The daily support call volume for Windows XP alone is roughly 2,600 calls per day, including Saturday and Sunday. Based on information provided by Dell to Microsoft in the ordinary course of business, Dell receives about 1.2 million support-related calls per month, divided roughly evenly between hardware and software issues.

55. With this background information in mind, consider the effect of Section 1 of the States' Remedy on the support of Windows operating systems. Continuing with the conservative illustration used above, there would now be more than a thousand variants of the operating system to support. The first support problem to be faced is determining which variant of the operating system that the customer has. Traditional means of helping customers, such as the Help system included with the operating system, will be of little use in the face of this variability (especially since the Help system itself is one of the "Microsoft Middleware Products" that Section 1 requires to be removable).

56. Section 1 of the States' Remedy thus creates an exponentially more difficult, and therefore intractable, support burden for Microsoft. Ms. Moore estimates that each operating system variant could lead to a two-fold increase in call volume, doubling Microsoft's support costs for each new variant. This would be particularly true if the variant was created by removing or modifying a critical part of the operating system, like the software in Windows that supports a major portion of the Win32 API set. Such removal or modification is possible under Section 1 of the States' Remedy, since the broadly defined "Middleware" includes such software.

57. Not surprisingly, Microsoft managers have observed a direct correlation between increased testing and lower support costs. Further, the cost of fixing a software bug roughly doubles at each step in the software lifecycle (Beta 1, Beta 2, RTM, etc.). Thus the less thorough testing of Windows that would result from Section 1 of the States' Remedy would increase support costs (and I would expect decrease consumer satisfaction as well, although this is not my area of expertise).

58. From the support standpoint, things only get worse when non-Microsoft middleware products replace the equivalent Microsoft software. Since Microsoft support personnel will not have detailed knowledge (or perhaps *any* knowledge) of the inner workings of non-Microsoft products, they will be unable effectively to respond to consumer need for support. It will be nearly impossible for Microsoft personnel to diagnose or correct problems that may stem from the behavior of software over which Microsoft has no control. These problems will be exacerbated as new versions of non-Microsoft middleware products emerge and as Microsoft operating system products evolve. Over time, Windows would become effectively unsupportable, resulting in serious negative impact on consumers.

5. Windows XP Embedded Does Not Demonstrate that Section 1 of the States' Remedy is Feasible.

59. Considerable attention has been given in this case to a Microsoft software product called "Windows XP Embedded" in the context of Section 1 of the States' Remedy. Statements made by various witnesses suggest that the existence of Windows XP Embedded demonstrates that Section 1 of the States' Remedy is technically feasible. In my opinion, these statements reflect a misunderstanding of what Section 1 requires or of the nature and purpose of the Windows XP Embedded product, as I explain below.

60. In order to develop a better understanding of Windows XP Embedded, I built working binaries from the Windows XP source code, observed how these binaries are included into the Windows XP Embedded product, and then built and booted working versions of Windows XP Embedded runtimes for a Sony PCG GR370 laptop computer that I configured to "dual boot" Windows XP and those Windows XP Embedded runtimes. I also conducted a telephone interview of Bruce Beachman, Product Unit Manager of Windows XP Embedded, on April 25, 2002. I have in part relied upon factual information provided by Mr. Beachman in the preparation of this section of my testimony.

61. The purpose of the Windows XP Embedded tool suite is to facilitate the creation of Windows XP Embedded runtimes that are tailored to specific hardware running specific applications. Examples of such embedded devices include network routers, point-of-sale terminals, ATM machines, information kiosks and the like. Windows XP Embedded runtimes are created by selecting the specific functionality required by the target application running on the target hardware. Embedded devices typically have very different requirements than personal computers. For example, many embedded devices do not require a keyboard, mouse, display (i.e., no interactive user interface) or writeable hard drive. Embedded devices also typically have less memory than personal computers.

62. To construct a Windows XP Embedded runtime, the embedded device developer selects the specific desired functionality using software tools designed for that purpose. The granularity of selection is at the level of Windows XP binaries (that is, the files of executable code that run on a personal computer) assembled into components defined by different feature teams throughout the Windows organization. There is no separate "source code" for Windows XP Embedded. Thus, Windows XP Embedded is simply a collection of Windows XP binaries selected for a specific target device and application. These binaries represent an inflexible "componentization" of the Windows XP operating system. This componentization is merely a reflection of boundaries already present within the Windows XP source code; it does not reduce, or otherwise alter in any way, the existing interdependencies found among the various components that make up that source code.

63. It may be helpful to review the process within Microsoft by which a Windows XP "feature" becomes a Windows XP Embedded "component". Individual features of Windows XP are developed and managed by teams of software developers called "feature teams". There are many such teams, comprising thousands of programmers, each responsible for developing and maintaining the portions of the Windows XP source code tree that falls within their purview. To support the Windows XP Embedded product, each of these teams builds an "SLD" file (also called a carrier file) that contains the component definition. This definition includes, among other things,

a list of the individual binary files that are logically included in a given component, as well as the Windows registry data required for these binary files to execute.

64. Although it is a complex process, I think it is useful to explain the way in which the components that make up Windows XP Embedded are tested to identify dependencies. After constructing the SLD file for a particular component, Windows XP feature teams utilize an internal tool called the "Component Verifier" to check the completeness and validity of that SLD file. The Component Verifier runs on standard Windows XP (not on a Windows XP Embedded runtime) and monitors system activity while the component (e.g., Internet Explorer) executes. The monitored activity includes the loading and unloading of modules (e.g., dynamically linked libraries or DLLs), as well as Windows registry activity that occurs during execution of the component. This is necessary because registry keys can create dependencies (for example, a PATH variable), in addition to the conventional dependencies created when one software module invokes (calls) functionality in another software module. Using the Component Verifier, the Windows XP feature teams can identify many, although not all, of the dependencies that exist between their component and other components of Windows XP Embedded.

65. Not all features of Windows XP become Windows XP Embedded components in this manner. A substantial amount of Windows XP software (by my count, 1,971 Windows XP software modules) is not relevant to embedded environments and is not included in the Windows XP Embedded tool suite. Also, the core of the Windows XP Embedded operating system (the "base component") is developed and managed not by a feature team, but by the Windows XP Embedded design team. The base component represents the minimum functionality to boot a working system, although that system has very little functionality – for example, it cannot run any Windows applications. However, as I have explained above, under Section 1 of the States' Remedy, even portions of this "base component" fall within the definition of "Microsoft Middleware Product" and therefore must be made optionally removable. This is an example of how the components identified in Windows XP Embedded do not even solve the problem of identifying which pieces of Windows operating systems would have to be made optionally removable under Section 1. In addition, nothing in the States' Remedy suggests that the component definitions that exist in Windows XP Embedded would be considered acceptable definitions for even a small subset of Microsoft Middleware Products.

66. Windows XP Embedded runtimes are created by embedded developers using the Windows XP Embedded Studio. The Windows XP Embedded Studio contains the following tools:

· Component Database: The component database contains the definitions of platforms, components and their supporting data. It is implemented as a SQL Server database. Component definitions are associated with a specific platform and reference resources that comprise the component's functionality. Component definitions also contain information about the logic required to add specific functionality to a Windows XP Embedded runtime. The Component Database does *not* contain the actual software binaries associated with the component definitions. These are kept in a separate repository.

· Target Designer: A tool used to customize a Windows XP Embedded runtime using components selected from the database, and then to assemble the actual runtime image.

· Component Designer: A tool used to create component definitions. Embedded designers can use this tool to create the application that their device needs in order to perform its intended function, for example, the software program that responds to a user's request to withdraw cash from her checking account at an ATM.

· Component Database Manager: A tool used to import component definitions into the Component Database.

· Platform-specific tools: Tools that assist with hardware analysis (such as the Target Analyzer), image deployment and data conversion (e.g., converting device driver INF files to components).

67. Specific Windows XP Embedded runtimes are built by OEM or ISV embedded developers as follows:

i. The hardware configuration of the specific target device must be analyzed, either manually or by using a software tool such as the Target Analyzer. The Target Analyzer operates by attempting to enumerate all of the plug-and-play devices present on the target device. To the extent that it is able to do so, the Target Analyzer produces a definition of the hardware configuration that can then be imported into the Component Designer or the Target Designer in order to select the appropriate hardware-specific components to be included in the runtime image. The hardware coverage provided by the Target Designer is not complete. In most cases, the components delivered with Windows XP Embedded will not be sufficient for a specific target device. For some hardware devices, new components will have to be developed, or existing Windows XP compatible device drivers will have to be componentized using a tool provided with the Windows XP Embedded tools suite for that purpose.

ii. The features and functionality required in the Windows XP Embedded runtime must be selected using the Target Designer. This includes the target application software (this is just another component in the context of Windows XP Embedded) that will execute on the Windows XP Embedded runtime under construction - for example, the software that responds to user requests for money at an ATM machine. In addition, all Windows XP API calls made by the target application software must be identified so that the necessary Windows XP Embedded components can be included in the runtime image. This is potentially a complex and time-consuming task. For example, a likely scenario for an embedded developer to obtain this information would be to employ third-party debugging software to "hook" all Windows XP API calls made by the target application, and then use the "filter" software provided by the Windows XP Embedded tool suite to search for the component that contains the Windows XP DLL file that contains support for this call. This procedure has to be followed for each of the Windows XP API calls made by the target application. Windows XP Embedded also allows for the selection of low-level system features such as the type of file system support (e.g., FAT or NTFS), as well as the inclusion of features like Windows Media Player or Internet Explorer.

iii. As noted above, a Windows XP Embedded runtime must be built by the embedded designer. Building a runtime image using Windows XP Embedded Studio differs from building an application from source code. Instead of creating a new image by compiling the operating system source code, Target Designer assembles an instance of Windows XP Embedded from its individual "components". The Windows XP Embedded build process is an iterative process, consisting of the following major steps: dependency checking and resolution, file and resource assembly, construction of the runtime image and creation of the necessary Windows XP registry hives (the registry entries required for components to execute correctly). Typically, it is necessary to go through this process many times before all other components necessary to enable the components selected by the embedded designer have been included in the final runtime image. Even then, extensive testing is necessary to ensure that the Windows XP Embedded runtime functions as intended, as I explain below.

68. The result of using the Windows XP Embedded tools suite in this manner is the creation of a single-purpose instance of the Windows XP Embedded operating system. By "single purpose" I mean an instance targeted to a particular configuration of a particular set of hardware running a particular (small, usually one) set of applications. An instance of a Windows XP Embedded runtime is not a general purpose operating system that can load new hardware or software functionality at will.

69. It is critical to an understanding of this process to observe that a Windows XP Embedded runtime created using the Windows XP Embedded Studio is untested. Each assembly of Windows XP Embedded components created for a specific application and target hardware is likely to be different. These instances must be extensively tested by the embedded developer (typically by full feature regression testing) prior to deployment in a limited-purpose device. For example, even if it were somehow possible to use the Windows XP Embedded tools suite to construct a fully functional general purpose instance of Windows XP (with some subset of features removed in support of Section 1 of the States' Remedy), this construction would have to be subjected to the full regression testing that any other Windows operating system receives. This testing would likely take as long (almost two years), and consume the same very large set of resources as any other Windows operating system.

70. It is also important, in my view, to remember that the process of constructing a Windows XP Embedded runtime is fundamentally *additive*, while the process envisioned by Section 1 of the States' Remedy is fundamentally *subtractive*. In other words, an embedded developer builds a runtime image by adding the components required to support his or her special hardware configuration and application, and then the Windows XP Embedded tools suite automatically pulls in all of the other components on which the components selected by the embedded designer rely. This is the exact inverse of removing random components from the operating system without regard to their dependencies on other components. Thus Windows XP Embedded provides no capability for removing components without impairing the performance or degrading the functionality of the balance of the operating system.

71. This important distinction is illustrated in Figure 4. As I will explain during oral direct testimony, this Figure depicts in a very simplified way the *additive* process by which XP Embedded can be used to build a single purpose operating system by selecting certain components

defined in Windows XP Embedded. The Target Designer tool then identifies other components that must be present for the selected components to work. The combination of components is assembled into a single purpose operating system. Figure 4 also depicts why Windows XP Embedded is not *subtractive*. If the Windows XP Embedded tools are used to assemble components into an operating system, but then one of those components is "removed" (for example because it is a "Microsoft Middleware Product" under the States' Remedy, which most or all XP Embedded components would be), all the functions of the operating system that are dependent on that component will not operate. Windows XP Embedded does nothing to solve this problem, which is a key reason why Windows XP Embedded does not demonstrate that Section 1 is technically feasible.

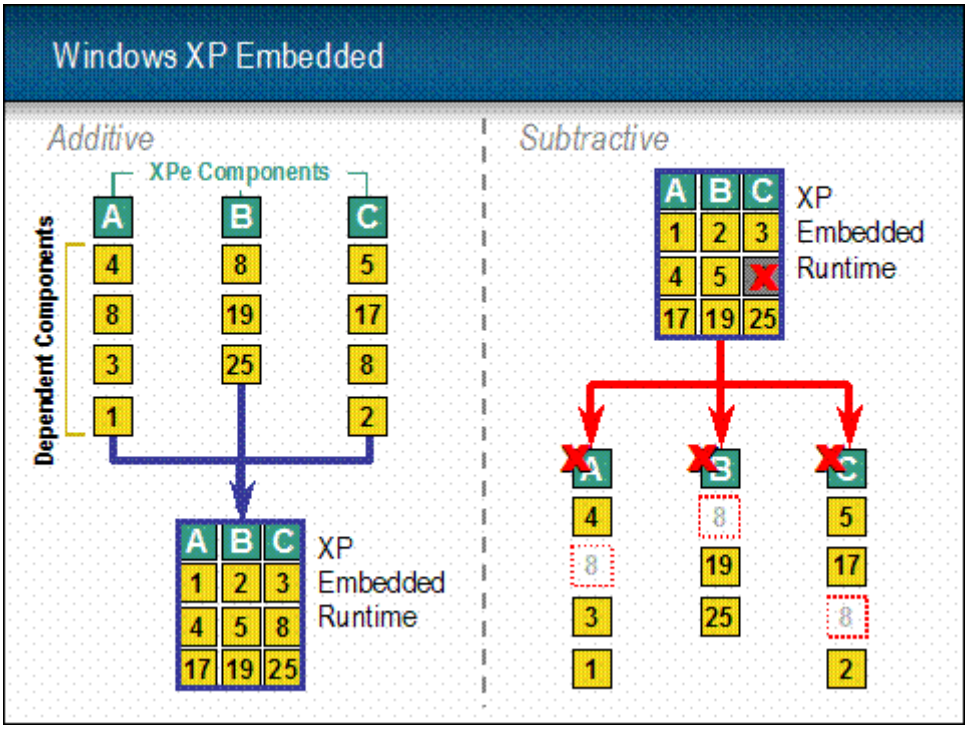

Figure 4

72. Further, Windows XP Embedded does not facilitate, or even permit, a division of software module functionality beyond that already present in the Windows XP binaries. It is not possible to improve, modify or even examine Windows XP binaries using the Windows XP Embedded tools suite.

73. I therefore conclude that the "componentization" offered by Windows XP Embedded does *not* support the optional removal of "Middleware" and "Microsoft Middleware Products" contemplated by Section 1 of the States' Remedy. In fact, using the Windows XP Embedded Studio has only confirmed my previous understanding that there are large numbers of cross-dependencies among the binaries that comprise Windows operating systems that would make it exceedingly difficult to pull out various instances of those binaries without causing serious malfunctions.

6. The Experiments of Dr. Edward Felten Do Not Establish that Section 1 is Technically Feasible.

74. In his testimony in this case, Dr. Appel appears to offer the prior work and testimony of Dr. Edward Felten as support for the technical feasibility of Section 1 of the States' Remedy:

"For example, in prior proceedings before the District Court in this case, Professor Edward Felten indicated that he was able to locate an interface between the Windows operating system and Microsoft's Internet Explorer browser. The presence of an interface indicates that the browser was developed as a separate module. Professor Felten also showed that for almost all purposes, the Internet Explorer module could be replaced by the Netscape Navigator module, because they both matched the same API." [16]

75. After examining the direct testimony of Dr. Felten, as well as the relevant trial transcripts and exhibits containing the source code of the software written by Dr. Felten and his assistants to carry out his experiments[17], I find myself in disagreement with Dr. Appel's conclusions on this issue. Although just what software constitutes "Internet Explorer" has to my knowledge never been defined in this case, it appears that what Dr. Appel refers to as the "Internet Explorer module" is in fact by my count a collection of about sixty-nine software modules within Windows. These sixty-nine software modules provide a wide range of functionality to the operating system, including, in addition to web browsing, HTML rendering, multiple screen support, the Windows Help system and local file system browsing, among other things. There is no standalone web browser that can be neatly excised from the operating system. Further, Internet Explorer was not "replaced" by Dr. Felten's "prototype removal program". In fact, other than minor changes to user preference items associated with Internet Explorer, Dr. Felten's prototype removal program replaced only two software modules associated with Internet Explorer with slightly modified versions, and added a single software module that exports only one function.

76. Dr. Felten's modifications to the URLMON.DLL module in Windows 98 cause the operating system to determine what software is calling Internet Explorer, and if that is IEXPLORE.EXE (the executable file for the Internet Explorer web browsing window), EXPLORER.EXE (the executable file for the Windows Explorer and My Computer browsing windows) or HH.EXE (the executable file for the Windows Help system), a message is displayed that there is no registered handler for the HTTP protocol in the operating system. Otherwise, all of the code associated with Internet Explorer executes normally. Thus, for example, if a third-party application such as Quicken calls Internet Explorer in order to display its HTML user interface or update the prices of stocks in the user's portfolio using HTTP to download such information from the Internet, the Internet Explorer code in Windows 98 is executed normally. This is because the executable file for Quicken is not one of the named processes that cause the message to be displayed. This demonstrates that Dr. Felten did not "remove" Internet Explorer and replace it with Netscape Navigator, as Dr. Appel suggested. The software code that supplies the functionality referred to as Internet Explorer is still present and still being relied on by other parts of Windows as well as by applications running on top of Windows.

77. It is interesting to note that Dr. Felten's version of Windows Update program takes advantage of a Microsoft Foundation Classes (MFC) Library call named

CHTMLVIEW::CREATE to invoke a web browser ActiveX control to display the Windows Update Website. This web browser ActiveX control exposes the Internet Explorer functionality of Windows to developers in an easy-to-use form. In his trial testimony Dr. Felten did not refer to his use of the web browser ActiveX control as a "web browser" because he did not invoke the available functionality that would have permitted his Windows Update program to display an Internet address bar. However, it would have been straightforward for Dr. Felten to add this behavior via another MFC call. This use of Internet Explorer functionality via the web browser ActiveX control clearly indicates that Dr. Felten did not "remove" Internet Explorer from Windows 98—he was using the Internet Explorer components of the operating system. In fact, Dr. Felten's reliance on the Internet Explorer functionality in Windows 98 demonstrates the benefit to developers of Windows applications of having that Internet Explorer functionality available in the operating system – Dr. Felten did not have to write his own HTML rendering engine and his own HTTP protocol support, which likely would have made his experiments considerably more time-consuming and difficult.

78. Finally, it is also interesting to note that Dr. Felten's first version of the modified URLMON.DLL resulted in what is called a "memory leak".[18] A memory leak refers to a typically catastrophic situation in which a program uses more memory than it keeps track of, usually resulting in program (or in this case, operating system) failure. Although Dr. Felten corrected this serious bug in the second iteration of his program, the fact that this problem could occur based on the sorts of minor modifications made to URLMON.DLL, even when those modifications were made by an experienced and knowledgeable programmer such as Dr. Felten, is another indication that changing an operating system as complex as Windows is quite challenging, and extensive testing is required to ensure that the resulting product will function as intended.

7. Dr. Andrew Appel's Suggested Methods for Microsoft to Comply with Section 1 of the States' Remedy Fail Upon Close Analysis.

79. In his redirected trial testimony, Dr. Appel suggested that Microsoft could comply with Section 1 of the States' Remedy in any of four ways:

"One way is to simply let the Microsoft middleware product be removable. Another way is to let subcomponents of the Microsoft middleware products be removable. The States' remedy doesn't require that, but it permits that. And then in the case of, for example, MS HTML, the rendering engine the subcomponent of the browser, an OEM might choose to leave that component in even if they want to substitute a different browser, and then there's no chance of degradation of the functionality of other components that depend on that HTML rendering. Another option, as I have explained, is to take necessary fragments of functionality and embed them in other products, other than Microsoft middleware products, so they don't expose APIs. Another kind of way to comply is just to reduce the inherent commingling, or I should say interdependence between the Microsoft middleware products."[19]

80. I will consider each of the options suggested by Dr. Appel for compliance with Section 1 in turn:

a. "let the Microsoft Middleware product be removable" – I have already discussed why the optional removal of the broad range of Windows components that the States' Remedy defines as "Middleware" and "Microsoft Middleware Products" is technically infeasible, cannot be done without degrading the operating system and leads to intractable testing and support problems.

b. "let subcomponents of the Microsoft middleware products be removable" – This option shares the same technical problems as the first option, and is potentially more problematic, since there are now even more operating system variants to test and support. The finer the granularity of components that are made optionally removable from the operating system, the greater the complexity of determining the cross-dependencies of those components (and consequently seeking to remedy the problems that will inevitably arise if those components are removed). In addition, in my view this option makes no sense because both such subcomponents and the remainder of any given "Microsoft Middleware Product" are themselves blocks of software code that must be made optionally removable under Section 1.

c. "take the necessary fragments of functionality and embed them in other products, other than Microsoft middleware products, so they don't expose APIs" – It appears that what Dr. Appel is suggesting with this option is that Microsoft replicate all of the functionality provided by "Microsoft Middleware Products" in portions of the operating system that are not "Microsoft Middleware Products", except that these other portions of the operating system cannot expose APIs. Here again, this does not make sense (in my view) for two reasons. First, there are very few parts of the operating system that are not "Microsoft Middleware Products" under the terms of the States' Remedy, so creating replicas of large numbers of components would create massive code duplication, increasing the size of Windows and likely impairing its performance. Second, I see no point in simply moving functionality around in the operating system (rather than "removing" it as Section 1 expressly requires) if that functionality is not exposed for use. Moreover, as I have explained previously, having multiple copies of the same functionality present in an operating system makes it more difficult to fix bugs relating to that functionality and increases the risk that those copies will diverge over time and give rise to incompatibilities. Finally, I do not understand how this option squares with Section 4, which requires Microsoft to disclose all of the interfaces of Windows operating systems and thus would appear to proscribe the "hiding" of useful APIs from ISVs.

d. "reduce the inherent commingling, or I should say interdependence between the Microsoft middleware products" – In this option, Dr. Appel appears to suggest that Microsoft modify its Windows operating systems so that components that fall within the definition of "Microsoft Middleware Products" are less interdependent. Given the sweeping definition of "Microsoft Middleware Product" found in the States' Remedy, I believe that this would require a complete rewriting of the Windows operating system. In addition to the monumental difficulty of such an undertaking, I seriously doubt it is possible to construct an operating system in which the States' Remedy's broadly-defined "Middleware" modules do not have significant interdependencies. In fact, all of my experience with operating systems suggests the opposite: namely, that designing an operating system in which each component is a stand alone island of functionality that can be removed without causing the balance of the operating system to degrade is neither desirable nor possible.

In summary, none of the options that Dr. Appel has suggested as ways for Microsoft to comply with Section 1 appears actually to comply with what the provision states. Either the options would cause various features of the operating system to break, which would degrade the functionality of Windows, or they would cause the operating system to bloat with redundant and potentially incompatible software code, which would impair the performance of Windows and the performance of applications that run on Windows. Either way, the "unbound" version of Windows exhibits "degradation". This confirms my opinion that compliance with the literal terms of Section 1 is technically infeasible.

C. Section 2 of the States' Remedy is Technically Infeasible.

1. Section 2 Exacerbates the Support Problems Created by Section 1 of the States' Remedy.

81. Although the majority of licensing issues are not within the scope of my expertise, three technical issues are raised by Section 2 of the States' Remedy. The first issue derives from the requirement placed upon Microsoft to license the plethora of operating system variants created by Section 1, and the granting of permission to Third-Party Licensees to resell these licenses. As described above, the commercial distribution of many versions that purport to be the same operating system creates an intractable support problem. In the event that a consumer experiences a problem, or even simply has a question, he or she will not know whom to call for support, nor will he or she (or whatever support organization is contacted) be able to identify readily the operating system variant that has been purchased. This issue will grow significantly more problematic over time as various "Middleware" components are upgraded or replaced.

82. It is also reasonable to expect that consumers will be confused and disappointed if they discover that what they thought was Windows – a product designed, tested and supported by Microsoft – was in fact an amalgamation that is partly Windows and partly other software, and for which no single entity is willing to take support responsibility. Dr. Appel's assertion that Microsoft could simply hang up on customers after learning that their version of Windows contained modules supplied by OEMs or Third–Party Licensees is in my view wholly inappropriate for commercial software products. Based on my experience developing and supporting software, customers expect software vendors to stand behind a product that carries a vendor's name and are unsympathetic to arguments that the problem lies with someone else.

2. The Definition of "End-User Access" Renders Section 2 Infeasible.

83. The second technical problem with Section 2 of the States' Remedy is found in Clause 2.c. This section (in Clause 2.c.iv), as was the case in Section 1, permits an OEM or Third-Party Licensee to both "remove the means of End-User Access for Microsoft Middleware Products" and "remove the code for Microsoft Middleware Products". Disabling the access mechanisms by which users knowingly invoke features of Windows operating systems (removing "the means of End-User Access for Microsoft Middleware Products") is on the surface a feasible means for an OEM or other Third-Party Licensee to give exclusive promotion to its choice of "Middleware" in its offerings. In fact, that focus on removing end-user access is the basic approach taken by Microsoft's Remedy.

84. However, “End-User Access” is defined in the States’ Remedy as “the invocation of Middleware directly or indirectly by an end user of a computer, or the end user’s ability to invoke Middleware”. The definition of “End-User Access” also includes “invocation of Middleware that the Operating System Product’s design requires the end user to accept.” All parts of this definition except direct invocation by the end user are technically problematic, as I explain below.

85. By including “indirect invocation” of “Middleware” in its definition of “End-User Access”, Section 2 of the States’ Remedy includes within the scope of removable components software that might be executed as part of the operating system performing useful work, but about which the user is completely unaware. This inclusion would permit the removal of potentially vital operating system code.

86. The second technical problem with this definition is that it extends to the use of “Middleware” by Windows itself, or the use of “Middleware” by application software developers (both Microsoft application developers and ISVs). As previously explained, a wide variety of products have been designed to rely on the APIs exposed by such “Middleware”, and they will fail to operate correctly (if at all) if this “Middleware” is absent from Windows. Thus, actually removing the code of a “Microsoft Middleware Product”, given the very broad definition of such products, will in many cases not be feasible. Further, there is no need to excise this code from the operating system in order to provide the remedy sought by the non-settling States: “to permit such licensees to customize Windows (including earlier versions of Windows[20]) to include whatever Microsoft middleware or competing middleware the licensee wishes to sell to consumers.” Such customization can be achieved by rendering carefully specified types of Microsoft Middleware invisible to the user, and by giving exclusive promotion to the desired alternative third-party software (for example, Netscape Navigator instead of Internet Explorer). This creates a technically reasonable end result in which, from the ordinary user’s perspective, the only “Middleware” present is the substitute “Middleware”, but in which potentially critical operating system and application functionality has not been lost. As I have noted before, this is the approach taken in Microsoft’s Remedy.

87. The third problem with the definition of “End-User Access” is its inclusion of “invocation of Middleware that the Operating System Product’s design requires the end user to accept”. All operating systems routinely invoke software that falls under the States’ Remedy definition of “Middleware” in the course of normal operation. Since this “invocation” occurs without any user input, the end user is thus “required” to accept the invocation. Attempting to somehow limit this perfectly ordinary operating system behavior would be catastrophic to the functioning of any operating system.

D. Section 3 of the States’ Remedy Creates an Unreasonable Support Burden.

88. Section 3 of the States’ Remedy requires that Microsoft support the immediate predecessor version of its operating system whenever a major new release of Windows is released. In my opinion, Section 3 places an unreasonable support burden upon Microsoft. Although the non-settling States did not define “support” in this context, their use of the “both directly and indirectly” qualifier implies a level of support consistent with or even identical to that of a current product. A plausible reading of the support requirements of Section 3 could imply on-going

product development, security updates, support for new and emerging input/output devices and protocols, service packs, etc. In order to provide this level of support, Microsoft would have to enlarge substantially its support organization and effort. For example, when a new feature was added to the latest Windows release, Section 3 would require that the new feature be added to the preceding Windows release as well. However, the preceding release of Windows may lack the underlying functionality to support adequately this new feature, thus necessitating the incorporation of the required support functionality into the preceding release.

89. Even a narrow reading of Section 3 (i.e., assuming that the predecessor operating system is not required to be kept completely up to date with respect to the current operating system) creates a significant support burden for Microsoft. Since the new operating system is likely to have at least as many variants as the old operating system, it is reasonable to expect that Section 3 will double the number of operating system variants that must be supported by Microsoft. Given the scale of Microsoft's existing support operation that I have detailed above, even doubling that operation would result in enormous expense.

E. Section 4 of the States' Remedy is Technically Unreasonable.

90. As a result of the application of the broad definitions underlying Section 4 of the States' Remedy, Section 4 effectively requires Microsoft to publish information describing the internal workings and all of the software interfaces of its Windows operating systems, and not just the Windows desktop operating systems addressed during the liability phase of the case. I will first summarize the sound technical reasons for operating system vendors *not* to make such information public; I will then address the specific problematic definitions and disclosure requirements related to Section 4.

1. There Are Sound Technical Reasons for Operating Systems Vendors to Decline to Make Internal Interfaces Public.

91. During the 1980s, the principles of data abstraction and modularity came to be recognized within the computer science community as offering programmers the ability to develop complex software that was easier to debug and maintain. Data abstraction refers to the technique of hiding internal data representations and exposing only a well-defined external interface for use by others. For operating systems, it was also common to expose two versions of this interface, a private interface for other software components that were "trusted", *i.e.*, known to behave in a particular manner, and a public interface for "untrusted" components that were not presumed to be so well behaved. Trusted components would typically be other parts of the operating system. Untrusted components would typically be third-party programs written to call upon operating system services, such as application and utility programs.

92. Applying the approach just described to operating system development, system calls that directly access critical operating system components would typically be part of the private interface; system calls that indirectly access critical operating system components in a restricted manner would typically be part of the public interface. One of the advantages of this approach to software development is that the internal details of data representation are hidden from the user of that data representation. The operating system developer is free to make changes to this internal

data representation as long as the external interfaces relied upon by third parties are preserved. There is also an advantage to having information about the private interface be closely held. If the number of modules using the private interface is small, it is easier to make changes in the private interface in order to improve system performance or reliability, or to add new functionality.

93. Once published, it is difficult to change the public interface used to access operating system services. This is because programmers who have written code that relies on that public interface expect it not to change. Significant changes to the public interface may even necessitate a complete rewriting of the original application program, a costly and time-consuming undertaking. In the case of operating systems developed by Microsoft, there are conservatively thousands of such application programs relied upon by millions of end users. Having such application programs malfunction would cause enormous support problems for both Microsoft and the vendors of affected application programs, in addition to inconveniencing large numbers of consumers.

94. A related concern is when some private function or data structure is discovered, made public and used by significant numbers of programmers. There are many individuals who delight in ferreting out these sorts of details, and the large numbers of books on the subject attest to the willingness of software programmers to take advantage of such information. Widespread use of some part of the private interface effectively moves this portion of the private interface into the public interface. The negative consequence of this practice is that the operating system developer now may be faced with a choice between improving performance, which requires a change to the private interface (and which the original designer may have had every intention of changing in the future), or maintaining compatibility with a de facto standard never intended as a permanent part of the operating system specification.

95. Well-disciplined application programmers do not use undocumented functions of an operating system. Following this policy is a good practice that benefits all concerned: (1) Microsoft, who is thereby free to improve the performance and functionality of its operating systems, (2) the application developer, who can now rely on his or her program working with future versions of the operating system and (3) the end user, who can upgrade his or her operating system without fear of breaking important applications.

96. Under the States' Remedy, there would effectively no longer be any private interfaces within the Windows operating system. Section 4 appears to mandate that all Windows interfaces be made public, with all of the deleterious effects that I have noted above.

97. I will now discuss the problematic definitions relevant to Section 4 of the States' Remedy.

2. The States' Remedy Definition of "API" is Overly Broad.

98. The States' definition of "API" goes far beyond any common use of that term. This definition is probably broad enough to include every procedure call made in a Microsoft operating system, regardless of whether or not the procedure call was designed or intended to be called by any software other than different code in the same operating system module. Exposing all "resources, facilities, and capabilities" of an operating system is not only not the customary

purpose of an API, it is unsafe from the standpoint of operating system reliability. Microsoft's Remedy calls for Microsoft to disclose any operating system calls made by clearly defined Microsoft "Middleware" that are not already part of the large suite of operating system services exposed to ISVs via published Windows APIs. From a technical standpoint, this is the appropriate and reasonable remedy.

99. Parts (b) and (c) of the States' Remedy definition of "API" appear to include all of Microsoft's server operating system internals. To my knowledge, server operating systems are not at issue in this case. Moreover, the disclosure of the details of security- and privacy- related components of the operating system, as called for in part (c), could adversely impact both individual and corporate users of Microsoft operating systems, who rely on that security and privacy in the normal course of managing their affairs (see Section V.K below).

3. The States' Remedy Definition of "Communications Interface" is Both Overly Broad and Technically Flawed.

100. In addition to its vast scope, encompassing virtually all computer-related technologies, the States' Remedy definition of "Communications Interface" fails to recognize the "public-private" interface dichotomy previously discussed. Nor does it take into account the potential threat to consumer privacy and security that would result from the disclosure of the internal details of security- and privacy-related components of the operating system, as discussed in Section V.K of my testimony.

4. The States' Remedy Definition of "ICP" is Overly Broad.

101. The States' Remedy definition of "ICP" is sufficiently broad so as to include anyone who maintains a web site. There are millions of web sites that have nothing to do with computer software or operating systems, including millions of web sites maintained by individual users. Although the States' Remedy definition is similar to Microsoft's Remedy definition, its breadth and usage has more impact in the States' Remedy, as discussed below.

5. The States' Remedy Definition of "IHV" is Overly Broad.

102. The States' Remedy definition of "IHV" is sufficiently broad so as to include companies that make computer cables, printer cartridges or computer tables for people's home offices. Although the States' Remedy definition is similar to Microsoft's Remedy definition, its breadth and usage has more impact in the States' Remedy, as discussed below.

6. The States' Remedy Definition of "Interoperate" is Overly Broad.

103. The States' Remedy definition of "Interoperate" goes well beyond what I would consider to be the customary usage of the term: the requirement that two programs can exchange and make effective use of each other's data. Supporting the "full features and functionality" of a software product, as called for in this definition, appears to imply the ability to clone the other software product with which the first product normally communicates, in this case Microsoft operating systems, "Middleware", and applications. However, interoperation, as that term is customarily

used, does not imply cloning. For example, it is one thing to say that the directories of two server operating systems can interoperate by synchronizing the information stored in the two directories. That is very different, however, from saying that the two directories have the "full features and functionality" of one another.

7. The States' Remedy Definition of "ISV" is Overly Broad.

104. The States' Remedy definition of "ISV" places no limitations on the scale of operation of the ISV, or any other qualification for that matter, thus requiring Microsoft to treat equally a Fortune 100 manufacturer of personal computers and an undergraduate student who writes software in his or her dormitory room. Although the States' Remedy definition is similar to Microsoft's Remedy definition, its breadth and usage has more impact in the States' Remedy, as discussed below.

8. The States' Remedy Definition of "OEM" is Overly Broad.

105. The States' Remedy definition of "OEM" includes computer assembly facilities, including those that add no technical content to the products that they assemble, and including assembly facilities located in off-shore areas that may be experiencing serious problems with software piracy. This definition also includes "OEMs" that do not employ Microsoft products.

9. The States' Remedy Definition of "Technical Information" is Overly Broad.

106. The States' Remedy definition of "Technical Information" appears to be broad enough to include all information about any Microsoft product. In fact, the amended States' Remedy has broadened this definition to include specifically applications that run on "Microsoft Platform Software".

107. Although the amended States' Remedy deleted the words "and/or implementing" from the phrase "Technical Information means all information regarding the identification and means of using and/or implementing APIs and Communication Interfaces that competent software developers require to make their products running on any computer Interoperate effectively", the very broad definitions of "Communications Interfaces" and "Interoperate" of the States' Remedy render this exclusion moot. The provision continues to require Microsoft to disclose information about how APIs and Communications Interfaces are implemented in Windows operating systems.

108. Microsoft would almost certainly have to provide all of its source code in order to meet the requirements of the sweeping definition of "Technical Information". For example, the States' Remedy definition of "Technical Information" includes (without limitation) a "reference implementation". A reference implementation is the source code for a software product that others use to clone the product, often on another platform. In the case of Windows operating system products, there is only one implementation, which is the commercial product in the marketplace. Thus, the mandatory provision of "reference implementations" in the definition of "Technical Information" would effectively require Microsoft to surrender the source code to its Windows operating systems to competitors. In the case of Office, the States' remedy proposal presents Microsoft with the Hobson's choice of surrendering the source code to either its

Windows or Macintosh implementations as the "reference implementation" required by this definition.

109. The States' Remedy definition of "Technical Information" also requires Microsoft to disclose the details of security and privacy related components of its software ("encryption algorithms and key exchange mechanisms"). As discussed elsewhere in my testimony, such disclosure could have direct adverse impact on both individual and corporate users of Microsoft software, who rely on the security and privacy provided by Microsoft to protect their personal and confidential data.

10. The States' Remedy Definition of "Timely Manner" Would Stifle Microsoft's Ability to Innovate Windows Operating Systems.

110. The inclusion of "Platform Software developers" in the States' Remedy definition of "Timely Manner" renders this definition unworkable. Software undergoing development often changes dramatically during the development process. "Platform Software developers" routinely write trial code intended to explore new ideas or test new functionality. This exploratory code is never intended to be part of a final product. Further, these new ideas or functionality may involve significant innovations, the disclosure of which would render it impossible to seek appropriate intellectual property protection. The requirement to disclose such information would stifle innovation within Microsoft, which would negatively impact consumers. The disclosure requirement imposed by this definition would also have a negative impact on application software developers, because they might use a software feature that never becomes part of a product, thereby lengthening their development cycle, or in the worst case, rendering their product inoperative. This definition also fails to recognize the "public-private" interface dichotomy previously mentioned. The negative impact of such failure is discussed in Section V.A.

11. The States' Remedy Definition of "Microsoft Platform Software" is Overly Broad.

111. Given the definitions upon which the definition of "Microsoft Platform Software" relies, the States' Remedy definition of "Microsoft Platform Software" appears to include almost every Microsoft product, from Windows to Office. It is my understanding that the liability phase of this case addressed Microsoft's position in desktop operating systems running on Intel-compatible personal computers, not on Microsoft's software products as a whole.

12. The States' Remedy Definition of "Operating System" is Overly Broad.

112. The States' Remedy definition of "Operating System" includes Windows CE (Microsoft's operating system for small portable and handheld devices), all of Windows server operating systems and all of their successors. To my knowledge, neither of these classes of operating system is at issue in this case.

13. The States' Remedy Definition of "Platform Software" is Overly Broad.

113. See "Microsoft Platform Software", above.

14. The States' Remedy Definition of "Web-Based Software" is Overly Broad.

114. The States' Remedy definition of "Web-Based Software" is unreasonably broad. In fact, the non-settling States have chosen to amend this definition to give it even more sweeping scope. In the context of modern operating systems, this definition includes any software on any computer connected to a network. This definition almost certainly includes SQL Server and Access (Microsoft's two primary database applications), Active Directory (the directory included in Microsoft's Windows 2000 server operating systems) and all software that provides security to Microsoft operating systems.

115. The States' Remedy definition of "Web-Based Software" is important to an understanding of the States' remedy because "Web-Based Software" makes up part of the definitions of "Bind", "Communications Interfaces", "Middleware", and "Microsoft Middleware Products", giving these derivative definitions significantly broader scope.

15. Section 4 of the States' Remedy is Excessively Broad.

116. Given the sweep of the definitions used, Section 4 of the States' Remedy is in my opinion excessively broad in scope. For example:

· Section 4 requires Microsoft to disclose (or broadly license on a royalty-free basis under Section 15) its intellectual property to essentially anyone that makes any hardware or any software product for use in or with a computer. The broad definitions offered in the States' Remedy would allow those with no reasonable need to have access to technical information related to Microsoft products to have such access, e.g., a company that manufactures printer ink-cartridges, or in fact anyone with a web site (an Internet Content Provider (ICP), according to the States' Remedy definitions). Once such broad-ranging intellectual property had been disclosed publicly, it would be relatively straightforward for anyone to clone Microsoft's products, including flagship products like Windows XP and Office XP.

· Section 4 appears to require that Microsoft make available the source code for *all* of its software that exposes APIs (because of the very broad definitions of "Microsoft Platform Software" and "Microsoft Middleware" contained in the States' Remedy). Software source code represents the "crown jewels" of any software company. The source code embodies all of the innovation, trade secrets and other intellectual property that forms the basis of the company's value.

· Paragraph 4.a.iii likely includes all software used to operate MSN, an online service that competes with AOL.

· As I noted above, the definition of "Technical Information" requires a "reference implementation" to be provided by Microsoft. A "reference implementation", although not defined in the States' Remedy, is typically the source code for some known implementation of a software product. In the case of Microsoft software, it is likely that the only available implementation to use for this purpose will be the commercial product from which Microsoft hopes to derive financial benefit. This definition effectively makes software sold by Microsoft

open-source. If instead the "reference implementation" is supposed to run on an operating system other than Windows (e.g., Linux), then Microsoft would be forced to develop versions of Windows operating system components that run on other operating systems.

· The definition of "Technical Information" contained in the States' Remedy is open-ended ("not limited to reference implementations …"). As a result, I do not see any basis on which Microsoft could resist a request for information about the internals of its products made by one of its competitors.

117. Microsoft need not publish a verbatim copy of its source code in order for Windows private interfaces to become public, with the resulting negative impact on Microsoft, software developers and consumers. This is because the ability to "study, interrogate and interact with the source code and any related documentation and testing suites" required by Clause 4.c would have the same effect. Further, under the terms of the States' Remedy, essentially anyone who owns a computer would likely be able to become a "qualified representative" of the very broadly defined set of "OEMs, ISVs, IHVs, IAPs, ICPs, and Third-Party Licensees". Further, I believe Dr. Appel is incorrect in testifying that it would be necessary to make a verbatim copy of large portions of the 38 million lines of source code that comprise Windows XP in order to misappropriate valuable intellectual property. In my experience, what is most significant about source code is not the way in which any particular piece of it is written, but instead the innovative ideas reflected in the source code. For example, a competitor with a keen interest in understanding how Microsoft implemented a particular feature like file encryption would only need to focus on that portion of the source code, and could learn a great deal even if the competitor did not make a verbatim copy of any portion of the source code.

118. Finally, I note that there are tens of thousands of third-party Windows applications the development of which was facilitated by the rich set of APIs that Microsoft documented through its MSDN and predecessor programs. These applications would not exist if Microsoft had not already provided and documented the APIs that these applications need. As a result, it is difficult to see what benefit would be derived from Section 4 that could possibly counterbalance its adverse consequences.

F. Section 5 of the States' Remedy is Technically Infeasible.

1. The Scope of Section 5 of the States' Remedy is Unreasonably Broad.

119. Section 5 of the States' Remedy would appear to require Microsoft to be aware of, acquire and test all "non-Microsoft Middleware" products for compatibility with Microsoft Platform Software. That is because Section 5 prohibits Microsoft from making changes to Windows operating systems that it "knows or reasonably should know" will "directly or indirectly interfere with or degrade the performance or compatibility of any non-Microsoft Middleware … other than for good cause." Given the huge number of software products being developed in the United States and abroad, and the difficulty of predicting how changes made to Windows will affect those software products, it is not clear how Microsoft would be able to satisfy this requirement. The large number of Windows operating system variants created by Section 1 of the States' Remedy only exacerbates this problem. The primary technical flaw with Section 5 is that Microsoft has no

knowledge of the inner workings of non-Microsoft software, and no way of obtaining this information short of exhaustive and inefficient testing.

120. In summary, Section 5 of the States' Remedy would impose an immense testing burden upon Microsoft. Resources devoted to such testing could otherwise be employed to develop new and improved versions of Windows.

2. Section 4 of the States' Remedy Exacerbates the Negative Effect of Section 5.

121. The broad disclosure requirements of Section 4 of the States' Remedy have an additional adverse effect in the context of Section 5. Because all software interfaces in Windows operating systems are required to be disclosed, even the "private" interfaces described above, it will be impossible to improve the performance or functionality of those Windows operating systems if such improvement would change the behavior of a system call, even a system call that Microsoft intended to remain private for that very purpose. This situation is likely to be present in many circumstances, and the resulting loss of flexibility would cause Windows to stagnate. Interfaces intended to be private (and thus subject to enhancement) would be forced to remain static, and the ability to add new features and functionality to the operating system would be impaired.

G. Section 10 of the States' Remedy is Technically Infeasible.

122. The broad definition of "Microsoft Middleware" contained in the States' Remedy (essentially any software module that contains a procedure called from any other module is "Middleware" under the States' Remedy definition) makes Section 10 technically unworkable. Many, if not most, of these "Middleware" components were not designed to be removed or replaced (or in many cases even visible outside of the operating system for the reasons described above). Thus the notion of "Default Middleware" as used in this context is inappropriate.

123. To make Windows support this sort of modular replacability of components would require a major redesign of the Windows operating system (assuming such a redesign is even feasible), the result of which would likely exhibit seriously degraded performance. This is because of the need to insert code that checks to see that every exposed system call is being correctly invoked. Even if such a serious engineering effort were undertaken, Microsoft would still have no way of ensuring that any replacement "Middleware" would have the necessary (and correct) functionality, short of exhaustive testing of every possible combination of Microsoft and non-Microsoft software. For the reasons previously enumerated, testing and support of these many variants of "Windows" poses an intractable problem.

H. Section 12 of the States' Remedy is Technically Problematic.

124. Although the non-settling States have significantly reduced the scope of what they consider to be "Browser" software in the amended States' Remedy, the definition remains problematic. This is because the definition of "Browser" in the States' Remedy states that "'Browser' means Internet Explorer 6.0, MSN Explorer 6.10, or their successors…". Although the technology called "Internet Explorer" appears to have been a central issue in this case, to my knowledge there is no definition in evidence of just what "Internet Explorer" means. Possible definitions range

from simply the top level executable file that launches an Internet Explorer web browsing window to all of the software in Windows that provide any of the functionality even loosely associated with web browsing. As computing evolves to a more web services–based computing model, the latter interpretation could come to mean almost anything that anyone wants it to mean.

125. The inclusion of the language that "Microsoft shall identify, provide reasonable explanation of, and disseminate publicly a complete specification of all APIs, Communications Interfaces and Technical Information relating to the Interoperation of such Browser software and each Microsoft Platform Software product" in Section 12 of the States' Remedy suggests to me that the non-settling States intend a very broad interpretation of "Internet Explorer" in this context, particularly given the very broad definitions of the terms "APIs", "Communications Interfaces", "Technical Information" and "Interoperate" already discussed. The specific exclusion of "the source code for non-Browser software that relies in whole or in part on functionality in Browser software" in Section 12 is a notable exception to this broad interpretation.

126. Definitions aside, Section 12 of the States' Remedy requires the wholesale disclosure of a significant body of Microsoft's intellectual property. Although I am not an economic or legal expert, I know, based on my own experience in the private sector, that the intellectual property embodied in software source code is one of the primary assets of any software company. I have relied on intellectual property protection to protect the value of software that I have developed, and I find it extremely troubling that any proposed remedy in this case would contemplate the apparent seizure of Microsoft's intellectual property in this manner.

127. Section 12 of the States Remedy also requires Microsoft to convey a license to "make, use, modify and distribute without limitation". This means that anyone could create an arbitrarily modified version of Internet Explorer, or any of the other technology that falls under the definition of "Browser", and label and distribute this product as the "real thing". The resulting confusion would negatively impact both consumers and Microsoft. For example, consider the effect on both the consumer and on Microsoft if someone modified Internet Explorer to no longer support Windows Help (the mechanism by which users can find answers to questions they have about the operating system) or Windows Update (the mechanism by which users can get updates and bug fixes, including security updates, to Windows software). If those features no longer worked, consumers would be worse off.

128. It is interesting to me that the States' Remedy expressly forbids Microsoft to require that any modified versions of Internet Explorer remain compatible with the version in Windows. Sun Microsystems imposes compatibility tests on Java licenses because it sees the dangers of platform fragmentation.[21] Even if one accepted the notion that Microsoft should be required to place large blocks of Windows source code in the public domain, it strikes me as perverse to include provisions in Section 12 that would increase the chances that variants of Internet Explorer would diverge, thereby generating more incompatibilities.

I. Section 13 of the States' Remedy is Technically Flawed.

129. The principal technical problem with Section 13 of the States Remedy is one of timing. Given the lengthy development and testing cycles of modern software, ninety days is an

unreasonably short period of time in which to expect Microsoft to integrate and test a newly-provided release of the Java runtime environment. Ninety days prior to a commercial release, an operating system vendor will be trying to limit any changes to fixing severe bugs. Microsoft is no different in this regard. To do otherwise could well jeopardize the release date. It is clearly in consumers' best interest for Java runtime software to be thoroughly tested as it will be used (i.e., with Windows operating systems), thus sufficient time needs to be allowed for this testing to take place. I understand that there are various other issues of a business nature that might arise from Microsoft being required to include someone else's software code in Windows, but those issues are beyond the scope of my expertise and I defer to others to address them.

J. Section 14 of the States' Remedy is Technically Flawed.

130. In order to gain a better understanding of the issues related to Section 14 of the States' Remedy and its predecessors, I conducted a telephone interview of Kevin Browne, the General Manager of Microsoft's Macintosh Business Unit, on January 22, 2002. I have in part relied upon factual information provided by Mr. Browne in the preparation of this section of my testimony.

1. Office for Macintosh is not a Port of Office for Windows.

131. Section 14 of the original States' Remedy appeared to reflect a misunderstanding of the manner in which Office for the Macintosh products are developed and maintained. This was perhaps due to changes in the Office development methodology that occurred around 1997. In the late 1980s and early 1990s, Microsoft's Office products (e.g., Word and Excel) were developed using a "core code" strategy. The Office product source code was partitioned into a machine independent part (about 80% of the code) and a machine-dependent part (about 20% of the code). Only the machine-dependent portion of the source code differed between the Windows and Macintosh versions of Office products during that time period. In 1997, this approach was abandoned so that Office for Macintosh could be more fully developed as a Macintosh-specific application. The source code trees diverged at that time.

132. Today, Microsoft maintains a separate Macintosh software source code tree and a separate Macintosh software development team (which is located in Mountain View, California, not Redmond, Washington). In addition to the differences between the target operating system platforms of the Windows and Macintosh teams, the Macintosh development team uses a different development platform (Code Warrior) from the Office for Windows team. The Macintosh software development team operates in a largely independent manner from the Office for Windows team. This independence is evidenced by the many Office innovations that are Macintosh-specific. These include, among others, drag and drop installation, application self-repair, the Entourage mail application, Image Effects picture editing tools, Word Data Merge Manager, Excel List Manager, PowerPoint Movies and true Macintosh OS X support, including full user interface "Aquafication" and support for the Quartz graphics layer. None of these application innovations involved "porting" Office for Windows innovations to Office for Macintosh. In fact early Office for Macintosh products that had a Windows-like look and feel were poorly received.[22]

133. Since 1997, Office for Macintosh products appear to have diverged in other ways from Office for Windows. Office for Macintosh has better support for QuickTime than for Windows Media formats. Office for Macintosh supports Palm-based handheld computers, but not PocketPCs (the Windows CE-based alternative to Palm). Office for Macintosh also uses the Apple operating system's dialog boxes and conventions, which a ported Office for Windows application would not use.

134. According to Mr. Browne, the primary reason for interaction between the Office for Windows and Office for Macintosh development teams is to ensure that strict file compatibility is maintained. It is a significant benefit to consumers of Office software if it is possible to open Office for Windows documents on the Macintosh and Office for Macintosh documents on Windows machines. The recipients of the Office source code conveyed by Section 14 would be under no obligation to maintain this compatibility benefit for consumers; in fact they would not be under obligation to maintain any compatibility with Microsoft Office products.

2. Section 14 of the States' Remedy Does Not Recognize Important Differences Between the Windows and Macintosh Office Products.

135. In summary, Office is *not* simply ported from Windows to the Macintosh. Further, imposing the porting requirements of Section 14 ("Microsoft shall commercially release the same number of major releases of Microsoft Office for Macintosh as are released of Microsoft Office for Windows, *with features consistent with Microsoft Office for Windows*" [emphasis added]) upon Microsoft would likely lead to Office for Macintosh products that would probably not meet the specific needs of Macintosh users.

136. Finally, I note that Office for Macintosh OS X was *not* a "Unix port" because Apple chose to maintain application compatibility between OS 8-9 applications and OS X. Office for Macintosh developers were able to take advantage of this compatibility during development of Office for Macintosh OS X.

3. Section 14 of the States' Remedy Would Likely Require Significant Disclosure of Windows Operating System Code.

137. Section 14 also requires that Microsoft provide "source code necessary to enable porting of the new version of Office to other Operating Systems". Given that (1) Office makes extensive use of the functionality of the Windows operating system, and that (2) the States' Remedy adopts the view that extensive and broadly defined disclosure is required to even "Interoperate", it would seem likely that the list of "necessary" source code would be extensive. Moreover, the requirement to disclose this source code is ongoing for ten years. The source code for any new feature of Windows used by a new version of Office would presumably have to be disclosed as well.

138. Finally, I am again troubled by the apparent wholesale seizure and sale of Microsoft's intellectual property that is contemplated by Section 14. I view this as particularly problematic given that, to my knowledge, Office has never been an issue in this case.

K. The Security Exemption of Microsoft's Remedy Offers a Technical Benefit to Users.

139. Dr. Appel and other Plaintiff witnesses in this case have challenged the security exemption of Microsoft's Remedy as attempting to create "security through obscurity". The idea behind this assertion is that security protocols and algorithms exposed to public and academic scrutiny will be "hardened" through an iterative process of being published, studied, "cracked", improved and republished. Further, these witnesses argue that secret protocols and algorithms may have hidden flaws that could be exploited by malicious attack.

140. From a purely academic point of view, I am in complete agreement with these assertions. However, the context in which the proposed security exemption will be applied is not academic. Millions of current users of Microsoft Windows operating systems rely on Windows security protocols and algorithms to protect the integrity and confidentiality of their data. To invite attack and misappropriation by publishing the internal workings of the security mechanisms used to protect these data is dangerous, and has the potential seriously to damage individual and corporate consumers. Microsoft should not be required to publish these mechanisms any more than a bank should be required to publish the plans and specifications of its vaults. Finally, I note that the National Security Agency, arguably the leading expert on computer security in the United States, does *not* publish its internal security protocols and algorithms.

VI. Other Technical Issues Raised By Plaintiffs' Witnesses

141. A number of technical issues related to certain of Microsoft's actions have been raised in the testimony of Plaintiffs' witnesses in this case. In many cases, this testimony mischaracterizes the relevant behavior of the Microsoft software at issue, and in some cases relevant related information was not included in the testimony. In the paragraphs that follow, I address a number of these issues.

A. John Borthwick (AOL)

1. Windows XP Embedded

142. While acknowledging that he is neither a software developer nor an engineer, John Borthwick, Vice President of AOL Advanced Services, asserted that Windows XP Embedded shares the same code with Windows XP, and that Windows XP Embedded allows developers to customize the operating system by choosing desired components.[23]

143. I have already discussed (in Section V.B.5) why Windows XP Embedded does not demonstrate that Section 1 of the States' Remedy is technically feasible. In short, the fact that embedded developers can use binaries from Windows XP Professional to create specialized runtime images for use in limited-purpose devices says nothing about the existence of cross-dependencies among those binaries that prevent them from being removed from the operating system without causing other parts of the operating system and/or applications running on top of the operating system to malfunction.

B. Richard Green (Sun Microsystems)

1. Java Platform

144. In his direct testimony, Richard Green, Vice President and General Manager of Java and XML Platforms for Sun Microsystems, offered the view that "The key parts of the Java platform – the programming language, the class libraries and APIs, the compiler, and the JVM – were each altered in Microsoft's implementation in ways that impaired and undermined the commercial appeal of the Java technology."[24]

145. While commercial appeal is not my area of expertise, I observe that programming language evolution is a natural and healthy process. For example, Exhibit D depicts a model of programming language evolution that was created by Éric Lévénez.[25] This Exhibit shows that Java itself derives from several ancestors, including Smalltalk and C++. It would be a bad thing if programming languages did not change over time to take advantage of new learning in the field of computer science and new opportunities created by improvements in hardware and software technology.

146. Further, the record with regard to the commercial appeal and compatibility of the Microsoft Java Runtime Environment (JREs), does not support Mr. Green's assertion. For example, in April 1998, *PC Magazine* reported the results of testing the Java compatibility of several JREs, including those offered by Sun, Microsoft and Netscape, running twelve "pure" Java applications, eleven of which had been certified by Sun.[26] All JREs are supposed to be able to run what Sun refers to as "pure" Java applications, however none of the JREs tested was able to run all of the test applications, including Sun's JRE running on a native Sparc/Solaris platform. Of all of the JREs tested, the Microsoft JRE was able to run the largest number of the tested applications. The conclusion I draw from this is that Microsoft's JRE was at least at that time actually a more faithful implementation of the Java specification than Sun's own JRE.

2. Java Applet Tag

147. Mr. Green also stated that certain important APIs exposed by Internet Explorer for interfacing with a Java Runtime Environment are not documented by Microsoft. He offered a single example: the way that Internet Explorer automatically launches the Microsoft JVM when it encounters an "APPLET tag" on a web page.[27]

148. In order to gain a better understanding of this issue, I conducted a telephone interview of Michael Wallent, the Microsoft Product Unit Manager for Internet Explorer, on April 24, 2002. I have in part relied upon factual information provided by Mr. Wallent in the preparation of this section of my testimony.

149. An APPLET tag is used by a web content author to embed a Java applet in an HTML web page. While it is true that the "APPLET tag" interface referred to by Mr. Green has not been disclosed by Microsoft, Mr. Green fails to mention that the APPLET tag is considered an outdated construct by the World Wide Web Consortium ("W3C") - the international standards organization responsible for many standards related to the World Wide Web. The APPLET tag became outdated (a "deprecated element") upon the release of W3C Organization's HTML 4.0

Specification (published on 18 December, 1997). The W3C defines a “deprecated element” as follows:

“A deprecated element or attribute is one that has been outdated by newer constructs. Deprecated elements are defined in the reference manual in appropriate locations, but are clearly marked as deprecated. Deprecated elements may become obsolete in future versions of HTML. User agents should continue to support deprecated elements for reasons of backward compatibility. Definitions of elements and attributes clearly indicate which are deprecated. This specification includes examples that illustrate how to avoid using deprecated elements.”[28]

150. In 1997, the APPLET tag was deprecated in favor of the OBJECT tag (also supported by Internet Explorer) because the OBJECT tag gives the web content author more flexibility. For example, while the APPLET tag can only be used to embed a Java applet in an HTML web page, the OBJECT tag can be used to embed a variety of different objects in an HTML web page, including Java applets, other Java components, plug-ins, ActiveX controls and images. The HTML 4 Specification contains two simple examples of how to use the OBJECT tag instead of the outdated APPLET tag. Usually this requires only very minor changes to the HTML used to embed the applet, as shown in the excerpt of the HTML 4.01 Specification shown below.

“DEPRECATED EXAMPLE:

In the following example, the APPLET element includes a Java applet in the document. Since no codebase is supplied, the applet is assumed to be in the same directory as the current document.

<APPLET code="Bubbles.class" width="500" height="500">

Java applet that draws animated bubbles.

</APPLET>

This example may be rewritten with OBJECT as follows:

<P><OBJECT codetype="application/java"

classid="java:Bubbles.class"

width="500" height="500">

Java applet that draws animated bubbles.

</OBJECT>”[29]

151. Of particular interest in this example is the use of the "java:ClassFile" OBJECT tag "classid" construct. In Internet Explorer (and other web browsers as well), different Java Virtual Machines have unique identifiers that indicate which JVM should be invoked. This gives web content authors complete flexibility to decide which JVM will be used to render their content. Further, the implementation of the classid="java:ClassFile" construct is a fully disclosed and completely "pluggable" mechanism that works on multiple platforms (including Internet Explorer for Windows, Internet Explorer for the Macintosh and Netscape Navigator).

152. This point is confirmed by internal Sun documents concerning the ability of Internet Explorer in Windows XP to invoke Sun's JVM using the APPLET tag on a web page. As an August 20, 2001 email from Graham Hamilton that is in evidence as Defendant's Exhibit 1025 states: "Our existing Java Plug-in works in XP exactly as before. It still supports the <OBJECT> tag to run applets inside of IE." Mr. Hamilton goes on to note that Sun is working on adding support for the deprecated APPLET tag, but he concludes his email by noting that "I think we want to discourage any impression that our existing functionality is broken in XP. It ain't."

C. Carl S. Ledbetter (Novell)

1. Windows 2000 Professional "digital signatures"

153. Dr. Carl S. Ledbetter, Senior Vice President, Engineering/Research and Development, and Chief Technology Officer of Novell, stated in his direct testimony that Windows 2000 Professional requires that requests for network services use Microsoft proprietary "digital signatures", which he said prevents Novell's network operating systems from responding to such requests.[30]

154. Although his testimony on this issue is lacking in specifics, the technology to which Dr. Ledbetter apparently refers relates to some sort of private communication between Windows 2000 clients and servers. As an initial matter, I note that it is routine for clients and servers built by the same company to communicate using private protocols. There is nothing unusual about that. However, the larger implication of Dr. Ledbetter's statement, namely, that Novell's NetWare is unable to interoperate with Windows 2000 clients, is to my knowledge inaccurate. On cross examination on this issue, Dr. Ledbetter was shown Defendant's Exhibit 1203, a Novell technical document describing how NetWare 6 Servers are able to interoperate with Windows clients, and he acknowledged that this was in fact the case.[31] This is consistent with my understanding that Windows 2000 clients can obtain a wide variety of network services from non-Microsoft server operating systems. In fact, we have precisely such computing networks running in the Computer Science Department at the University of Colorado, and we are not prevented from achieving interoperability by any lack of "digital signatures" from Microsoft.

2. MAPI Disclosures

155. In his direct testimony, Dr. Ledbetter asserted that Microsoft did not disclose information regarding its modifications to its Messaging Application Programming Interface (MAPI) protocol, and consequently, non-Microsoft middleware developers cannot interoperate with Microsoft's version of MAPI.[32]

156. In order to gain a better understanding of the issues related to Dr. Ledbetter's assertions on this topic, I conducted a telephone interview of Steve Wells, the Software Development Engineer and Group Development Head for the Microsoft Outlook email and collaboration product that is part of Microsoft Office, on April 29, 2002. I have in part relied upon factual information provided by Mr. Wells in the preparation of this section of my testimony.

157. MAPI is a protocol designed by Microsoft and used for communication between the Microsoft Outlook client and the Microsoft Exchange server. As these client and server products have evolved over time, so too has the MAPI protocol evolved. In fact, there are now at least four protocols used by Microsoft and others in the personal information management (PIM) segment of the software business. These include:

· Microsoft Extended Messaging Application Programming Interface (MAPI, or Extended MAPI) – MAPI is an extensive set of functions that developers can use to create email-enabled applications. The full function library is known as MAPI 1.0 or Extended MAPI. Extended MAPI allows a server-based messaging system to exercise control over the messaging system on the client computer, including creation and management of messages, and management of the client mailbox, service providers, etc.

· Simple Messaging Application Programming Interface (Simple MAPI) – Simple MAPI is a subset of 12 functions in MAPI that enable developers to add basic messaging functionality to Windows-based applications. Simple MAPI includes functions to support sending and receiving messages, including the ability to log onto the messaging system, compose new messages, add and resolve recipients, send messages and retrieve and read messages from the inbox.

· Collaboration Data Objects (CDO) – Collaboration Data Objects ("CDO") was originally called "OLE Messaging" and later "Active Messaging". CDO is a Component Object Model (COM) wrapper for the MAPI library. CDO implements most but not all MAPI functionality, although far more functionality than is included in Simple MAPI. Activities that can be accomplished using CDO include:

o logging onto the messaging system with specific profiles or with anonymous authentication,

o composing messages, addressing and resolving recipients, sending, receiving, and reading messages, adding attachments, automating replies,

o managing calendars, creating meetings and appointments,

o managing folders and messages within the information store, and

o managing addresses, especially within the Personal Address Book ("PAB").

There are two distinct "flavors" of CDO, the MAPI-based CDO.DLL and the Simple Mail Transfer Protocol (SMTP) based CDONTS.dll.

· Web-based Distributed Authoring and Versioning (WebDAV) – WebDAV is a set of extensions to the HTTP protocol that allows users collaboratively to edit and manage files on remote web servers based on international standards.[33] Using WebDAV protocol methods, software developers can create, copy, delete, move or search for resources in the Microsoft Exchange store, as well as set and search for resource properties.

158. All of these protocols are fully described (including examples of their use) in the Software Development Kits ("SDK") available on the Microsoft Software Developer Network ("MSDN") web site, as well as through a subscription service. I regularly consult MSDN and find it a very comprehensive and useful source of information about Microsoft products, including various Windows operating systems.

159. As MAPI and other related protocols have evolved over time, third party software developers have had to expend engineering efforts to keep pace. Hewlett-Packard, Lotus and Samsung are examples of companies that have been able successfully to build client applications that interoperate with Outlook.[34] Microsoft has been justifiably reluctant to publish the "business logic" (for example, checking to see if a new appointment conflicts with an existing appointment) associated with Outlook, because such innovations are what enable Microsoft to distinguish its products from those of its competitors. Nevertheless, Microsoft appears to have demonstrated a willingness to work with MAPI service provider developers to ensure compatibility between their software and Outlook.

3. Microsoft's NetWare Client

160. Dr. Ledbetter also asserted that "Microsoft's version of the Novell client is based on IPX (Inter-network Package Exchange), an older and outdated networking protocol, instead of IP (Internet Protocol), the format used for all current and new products. Thus, customers using the Microsoft version of the client cannot use certain Web services and other newer network services that are IP-based." Dr. Ledbetter also asserted that "the Microsoft version of the Novell client does not support current Novell name resolution providers, which are software devices that point the user to various network services. Thus, many network services are not available to customers using the Microsoft version."[35]

161. It is true that the Novell NetWare Client developed by Microsoft is less capable than the NetWare client developed by Novell. This is not surprising, since NetWare is Novell's product. I would expect that having developed NetWare, Novell would be better suited than Microsoft to develop client software that shows off the features and functionality of Novell's network operating

system. Moreover, the extensibility mechanisms in the Windows user interface enable Novell to integrate NetWare servers into the login mechanism, file dialog boxes and other user-visible aspects of Windows.

162. Novell's business model is apparently to sell the NetWare server to corporate customers, but to make the client NetWare software freely available for download. These corporate customers are sophisticated, and are likely to have IT staff capable of downloading and deploying the NetWare client using some form of automated distribution. For example, there are numerous products like Novell's own ZENworks that can be used to install the NetWare client on thousands of computers in a large computing network without the need to access physically any of those computers. Thus the practical impact of the issue raised by Dr. Ledbetter is likely to be quite small.

4. Multiple UNC (Universal Naming Convention) Provider

163. Dr. Ledbetter asserted that the Multiple UNC (Universal Naming Convention) Provider ("MUP") in Windows NT 4.0, which is Microsoft proprietary code that facilitates communication between Windows clients and various servers in a network, degraded the performance of rival server operating systems such as Novell's NetWare. Specifically, Dr. Ledbetter complained that the MUP created the illusion that Microsoft servers could handle requests for remotely-stored files much faster than Novell servers.[36]

164. The MUP is responsible for locating the correct network provider for UNC connections. To perform this task, it "asks" every network provider installed in Windows if it is able to handle a given UNC connection request. The origin of the problem was that the original version of the MUP waited to hear back from every network provider before it decided what network provider should actually handle the network connection. After all redirectors responded, the MUP chose (based on response and priority) which redirector the application requesting a file would use to retrieve that file from the relevant server. Therefore, even if a resource was readily available and accessible over one redirector, the request was still made over the other installed redirectors before the request was completed.

165. This problem was not specific to non-Microsoft software. Microsoft network service providers could potentially experience similar delays to those experienced by NetWare. The problem appears to have been identified by Microsoft in August, 1996. A work-around was published in May, 1997, and to the best of my knowledge the problem was permanently resolved with the release of Windows NT 4.0 Service Pack 4.0 in September, 1998. I have seen no evidence that the problem was anything other than a bug of the type that often arises in the design of software products as complex as Windows NT 4.0.

5. Undocumented APIs

166. Finally, Dr. Ledbetter asserted that "Microsoft has allowed its own programmers and developers to access and rely upon unpublished APIs, calls, or other interface information to assure full interoperability of its products, while forcing competitors to use only limited sets of information that allow for 'interoperability,' but only in inefficient and constrained ways."[37]

167. Presumably Dr. Ledbetter is referring to the use of undocumented APIs by Microsoft application programmers, since operating system developers routinely use private interfaces for good technical reasons (as I have already discussed at some length). Michael Tiemann of Red Hat made this same allegation in the context of Microsoft Office developers, without providing any specifics regarding the alleged use of such undocumented APIs.[38] Dr. Ledbetter also does not identify any particular API in Windows used by Microsoft application programmers (his two examples, the MUP and Kerberos, are not relevant to this assertion), nor do the three references he cites identify any such use. The June, 2000 *ZDNet AnchorDesk* article "APIs: Microsoft's Hidden Full Nelson" by Jesse Berst[39] identifies no specific use of undocumented APIs in Windows by Microsoft application programmers. Further, the books by Sven Schreiber (*Undocumented Windows 2000 Secrets: A Programmer's Cookbook*)[40] and Prasad Dabek, Sandeep Phadke and Milind Borate (*Undocumented Windows NT*)[41] discuss internal APIs within Windows operating systems, not the use of undocumented Windows APIs by Microsoft application programmers. I have read both of these books, and I found no mention of internal API usage by Microsoft application programmers.

168. Although I have no knowledge of undocumented API usage by Microsoft application programmers, I would not be surprised if there were indeed some isolated examples of such use in some Microsoft products. I suspect that all software developers have been guilty of taking short cuts from time to time. However, it is apparently Microsoft's policy to discourage the use of undocumented APIs in Windows,[42] and the effect of implementing Microsoft's Remedy will be to disclose any use of such undocumented APIs by components of Windows falling within the definition of "Microsoft Middleware".

D. Steven McGeady (formerly of Intel)

169. Steven McGeady, formerly a Vice President of Intel, made several technical assertions about actions taken by Microsoft.

1. GDI Interfaces

170. Mr. McGeady alleged that Microsoft's inadequate documentation of Graphics Display Interfaces ("GDI") in Windows operating systems complicated and prolonged the development of Intel's VDI/DCI software.[43]

171. Intel's VDI software was intended to bypass (i.e., replace) the Windows GDI, thus providing access to the accelerated functions offered by newer graphics hardware, while maintaining application software compatibility with the Windows GDI. This was a technically challenging undertaking. Writing software that manipulates hardware directly, while preserving the state maintained by other software that manipulates the same hardware, is in my experience one of the most challenging of device driver development problems. Anyone attempting such a task should expect significant difficulties.

172. In this specific case, it is not surprising that the internal details of Microsoft device driver/hardware communication had not been disclosed. Microsoft had no reason to believe that anyone outside of Microsoft would have any reasonable need for this information. For example, if

today I wanted to bypass an nVidia graphics adapter device driver in the manner contemplated by Intel, I would not expect to have the internal details of the nVidia device drivers made available to me. Rather, I would expect to have to do a significant amount of reverse engineering. Apparently, Intel faced a similar challenge in developing its VDI software, and Microsoft apparently helped Intel overcome it.

2. API Disclosures

173. Mr. McGeady also stated that Microsoft's inadequate disclosure of APIs relied upon by Intel's Indeo Video software complicated and prolonged the development of this Intel software.[44]

174. Here again, Intel was undertaking to develop very low-level software that would have to be compatible with existing higher-level software, and they found that certain internal details of Microsoft device driver/hardware communication had not been disclosed. Mr. McGeady appears to acknowledge this situation in his direct testimony: "Some of these interfaces were sufficiently documented; some were documented, but had not anticipated our needs (asis typical in the case of innovative uses); and some interfaces were not documented altogether." [45] As with Mr. McGeady's complaint about GDI, I would be surprised if Microsoft had disclosed the information Intel was apparently interested in obtaining because Microsoft had no reasonable expectation that third-parties would be seeking to insert software in between Windows and the hardware in the way Intel was attempting to do.

3. Microsoft Outlook/Exchange Protocols

175. Mr. McGeady stated that Microsoft Exchange uses Microsoft proprietary protocols to interoperate with Outlook, e.g., the calendaring function, and consequently non-Microsoft email products cannot employ this calendaring function.[46]

176. I have already discussed this issue in the context of Dr. Ledbetter's assertions. Lotus iNotes Access for Outlook is one example of a non-Microsoft email product that, in fact, uses these protocols. As stated in the *iNotes Access for Microsoft Outlook Specification Sheet*, "The Microsoft Outlook user experience is unchanged with iNotes Access for Outlook; users simply work with their mail, calendar and task data on Domino instead of Microsoft Exchange. Familiar Microsoft Outlook features are supported, including rich text, folders, and integration with Microsoft Office applications."[47] Plainly, Lotus was able to do what Mr. McGeady has suggested is not possible.

4. Handheld Device Synchronization API Disclosures

177. Finally, Mr. McGeady stated that Microsoft does not disclose the protocols necessary to synchronize a handheld device with a Windows desktop PC or to authenticate a handheld device to Windows NT server.[48]

178. I believe that Mr. McGeady's assertion is inaccurate. The Pocket PC 2000 SDK, freely available from Microsoft's MSDN web site since April 2000, appears to have complete

information about ActiveSync (Microsoft's handheld device synchronization software). In addition, there is a book describing how to build ActiveSync Service Providers. Quoting from the developer technical documentation that Microsoft provides on its web site concerning the Pocket PC SDK:

"In the Pocket PC SDK, for the first time Microsoft is shipping the ActiveSync® SDK, documentation, a debugging tool, and even a sample for developers who want to write their own ActiveSync Service Provider. However, do not get over-excited. Writing an ActiveSync Service Provider is one of the toughest tasks in the development environment of Windows CE. You can find a complete step-by-step tutorial in my book 'The Windows CE Technology Tutorial' by Chris Muench, published by Addison-Wesley (ISBN 0-201-61642-4)."[49]

179. I have downloaded this SDK and confirmed the presence of ActiveSync Service Provider instructions and examples. Using this information, a skilled software programmer could do what Mr. McGeady has suggested could not be done.

E. Larry Pearson (SBC Communications)

1. Alleged Kerberos Use in Passport

180. Larry Pearson, Associate Director of Product Design, Strategic Marketing for SBC Operations, asserted that Microsoft has extended "the Kerberos security standard used in Passport that still today does not allow non-Microsoft servers to fully and securely interoperate with the modified version."[50]

181. In order to gain a better understanding of issues related to Passport, I conducted a telephone interview of Hal Howard, Microsoft General Manager for Passport, and Yordan Rouskov, Microsoft Developer Lead for Passport, on April 25, 2002. I have in part relied upon factual information provided by these individuals in the preparation of this section of my testimony.

182. Mr. Pearson's assertion on this issue appears to be based upon a misunderstanding of the Passport security model. Passport does not currently use Kerberos as its security protocol. While the current version of the Passport security protocol (sometimes referred to as the "tweener protocol") employs symmetric key ticket-based security, this protocol is not derived from Kerberos. Plans are underway to support both Kerberos and the Radius dial-up protocol used by ISPs in the next major release of Passport (Version 2.5). However, Passport will continue to support the tweener protocol as well.

183. Mr. Pearson also appears to be mistaken in his assertion that non-Microsoft servers are unable to interoperate with Passport.[51] While only a Microsoft Passport Server can be what is called a "Passport Domain Authority" (since Passport is a Microsoft-provided service), Microsoft publishes detailed information (in an SDK) on how non-Microsoft Passport implementations can be developed. Table 1, obtained from records maintained by the Passport group at Microsoft, enumerates thirteen non-Microsoft commercial Passport servers.

OS	WebServer	Company Name

Solaris	Apache	FTD
Solaris	Apache	Ofoto
Linux	Apache	eAccess PC to Phone
Solaris	Apache	Campmor
Linux	Apache	BuyItOnline
Linux	Apache	Duet GP (Pressplay)
Linux	Apache	Konica Online Lab
Linux	Apache	eAccess PC to Phone
Linux	Apache	eAccess PC to Phone
Linux	Apache	My Virtual Model
Linux	Apache	JPassport
Linux	Apache	NorwaySports.com
Solaris	Netscape	DicksSportingGoods.com

Table 1: Examples of Non-Microsoft Passport Servers

F. David Richards (RealNetworks)

1. Internet Explorer Media Bar

184. David Richards, Vice President, Consumer Systems for RealNetworks, testified that the Internet Explorer media bar has an undisclosed technical file association registry that allows it to override the regular registry entries that reflect users' choices of which software they want to use as the default player for various multimedia file formats.[52]

185. Mr. Richards appears to be correct in his assertion, but there are reasonable technical grounds for Microsoft to have made this design decision. Microsoft has no way to know that a RealNetworks media bar would behave in the same way as the Internet Explorer media bar if RealNetworks or some other software vendor attempted to replace the media playback functionality in the Internet Explorer Media Bar. Thus, to ensure a consistent user experience, Microsoft did not design the Internet Explorer media bar to permit its media playback functionality to be replaced with such functionality supplied by another vendor. However, ISVs are free to create their own media bars for use with Internet Explorer, as RealNetworks has done.

G. Jonathan Schwartz (Sun Microsystems)

1. ActiveX Controls

186. Jonathan Schwartz, Sun Microsystems' Chief Strategy Officer, asserted that because ActiveX controls are built using Microsoft's COM architecture, Microsoft's technical disclosures are insufficient to permit most ActiveX controls to run on non-Windows operating systems.[53]

187. The Mozilla ActiveX Project appears to contradict Mr. Schwartz's assertion on this issue. The Mozilla ActiveX Project has developed a free open-source plug-in that allows developers to embed ActiveX controls in any Netscape Plug-in API application. This plug-in hosts ActiveX controls allowing them to be used in browsers such as Netscape 4.x/6.x, Mozilla and Opera.[54]

2. Extensions to HTML

188. Mr. Schwartz also asserted that many web pages will not display correctly when using Navigator because only Internet Explorer recognizes various undocumented Microsoft extensions to HTML.[55]

189. HTML is an evolving standard (currently at Version 4.01). Each new version of the HTML standard has included new features. Both Microsoft and Netscape have, over time, developed extensions to HTML, many of which have become part of the HTML standard. My experience is that many browsers have difficulty rendering some HTML constructs. Different versions of different browsers are capable of rendering different HTML extensions.[56] I have many times encountered web pages that displayed disclaimers indicating that only Netscape Navigator could display the content on that page correctly. This sort of situation is normal with any rapidly evolving technology, and does not suggest to me any effort on Microsoft's part to create intentional incompatibilities with other web browsers. Rather, it is a byproduct of the competition between Netscape and Microsoft to make HTML more useful for web content developers.

H. Michael Tiemann (Red Hat)

1. SMB/CIFS

190. Michael Tiemann, the Chief Technology Officer of Red Hat, testified that Microsoft has manipulated the SMB/CIFS protocol in order to frustrate the ability of competing server vendors to provide file-sharing services to Microsoft clients.[57]

191. The record on this issue does not appear to support Mr. Tiemann's assertion. A group of about 25 people (about 8 of whom are active contributors at any given time) comprise a loose-knit community of open source developers known as SAMBA. This is also the name of their product, which supports both SMB and CIFS (all versions). In fact, there are published performance results indicating that at least in some cases, SAMBA outperforms Windows 2000.[58]

192. The SAMBA web site has a list of Frequently Asked Questions (FAQs) concerning SMB and CIFS protocols.[59] That FAQ states: "The net effect is that Microsoft is now documenting large parts of their Windows NT fileserver protocols. The security concepts embodied in Windows NT are part of the specification, which is why Samba documentation often talks in terms of Windows NT. However there is no reason why a site shouldn't conduct all its file and printer sharing with CIFS and yet have no Microsoft products at all."

2. TDS

193. Mr. Tiemann also asserted that Microsoft has not disclosed the TDS protocol and thus prevented Linux desktop PCs from interoperating with SQL Server, Microsoft's large database application for use on servers.[60]

194. Again, the record on this issue also does not appear to support Mr. Tiemann's assertions. A product known as "FreeTDS", for example, can be configured to support Microsoft's SQL Server. As the product documentation for FreeTDS expressly states:

"Does FreeTDS support MSSQL?

Yes, as long as you configure the libraries with the 4.2 or 7.0 protocol version. See the User Guide for details." [61]

In light of this statement, I do not see a basis for Mr. Tiemann's assertion that Linux desktops are prevented from accessing Microsoft's SQL Server. As with many of the other allegations made by the States' witnesses, achieving interoperability required some work to be done to implement a solution that was in part facilitated by the extensive information disclosed by Microsoft about its products. In this case, the product at issue is a server application, not even an operating system.

3. IMAP Extensions

195. Mr. Tiemann asserted that Microsoft has extended the IMAP standard for email services to defeat the ability of non-Microsoft email servers to provide full functionality to Microsoft email clients.[62]

196. Mr. Tiemann is apparently referring to MAPI, which I have already discussed in the context of Dr. Ledbetter's and Mr. McGeady's assertions.

4. LDAP and ADSI

197. Mr. Tiemann asserted that Microsoft has extended the LDAP protocol in an effort to prevent non-Microsoft servers from providing directory services to Microsoft clients. Mr. Tiemann also asserted that the Active Directory Service Interfaces only support programmatic access to Active Directory.[63] I believe both of these assertions to be incorrect.

198. Active Directory is the directory service included with Windows 2000 Server.[64] Simply put, a directory service is a mechanism for finding and interrogating network resources based upon their attributes. Almost anything can be a resource, e.g., a user account, a service, a printer or a computer. The Lightweight Directory Access Protocol ("LDAP") is an IETF Standard (RFC2251) for accessing directory services. Microsoft appears to provide support for both LDAP version 2 and version 3 in Active Directory.

199. In addition to supporting the industry standard LDAP protocol, Microsoft developed a higher-level and easier-to-use programming interface called Active Directory Services Interface or ADSI. ADSI is a set of interfaces that abstract the capabilities of directory services from different network providers to present a single view for accessing and managing network resources.[65] Windows application developers can use ADSI services to enumerate and manage resources in a directory service, no matter which network environment contains the resource. That means that programs running on Windows 2000 can use ADSI to access information stored in Sun's iPlanet Directory Server on Solaris or Novell's eDirectory running on NetWare.

200. It appears to me that Microsoft has gone to considerable lengths to make the functionality of Active Directory available to non-Microsoft clients and servers. In addition, Microsoft appears to have designed ADSI to provide access not only to Active Directory but to other non-Microsoft directory services as well. Microsoft has not disclosed sufficient information to clone Active Directory, nor would I expect them to do so.

5. Windows Code Pages

201. Finally, Mr. Tiemann asserted that Microsoft's adoption of the proprietary Windows Code Page creates interoperability problems for non-Microsoft operating systems when trying to render text. In particular, Mr. Tiemann alleges that Microsoft does not support the "Latin 1 ISO 8859-1" character alphabet.[66]

202. To the best of my knowledge, this allegation is incorrect. Windows operating systems appear to support a variety of code pages and character sets including Windows, OEM and ISO 8859. The Latin 1 ISO-8859-1 character set is among the many character sets supported in Windows operating systems. [67]

* **

I declare under penalty of perjury that to the best of my knowledge the foregoing is true and correct.

Executed in Boulder, Colorado on May 2, 2002.

John K. Bennett

[1] For readability, I have adopted the same nomenclature for these documents as that used by Dr. Andrew Appel, the States' technical expert, in his written direct testimony.

[2] My resume is attached as Exhibit A.

[3] Although Windows 98 and Windows 98 Second Edition are specifically excluded from the application of Section 1 of the States' Remedy, Windows 95 is not so excluded. To the best of my knowledge, Windows 95 is still widely distributed by Microsoft through the MSDN program.

[4] Section 1 of the States' Remedy.

[5] Paragraph 22.x(i) of the States' Remedy.

[6] Paragraph 22.w of the States' Remedy.

[7] Paragraph 22.x(ii).2 of the States' Remedy.

[8] Plaintiff Litigating States' Remedial Proposals at 2.

[9]Direct Testimony of Bill Gates, p. 30.

[10]*Id.*

[11] *Id.*

[12] The precise number of "Microsoft Middleware Products" is not important to this argument. The testing and support burden becomes intractable even if there are only a small number of new operating system variants.

[13] Transcript of Trial Record Before the Honorable Colleen Kollar-Kotelly, Volume 15, p. 3130.

[14] This was also the approach taken in the original remedies proposed by the United States and the eighteen Plaintiff States before Judge Jackson.

[15] The number of out-sourced support personnel varies year to year, but is especially high during product launch. Fifteen hundred people represents an average figure.

[16] Direct Testimony of Dr. Andrew W. Appel, pp. 50-51.

[17] Testimony of Edward W. Felten (including appendices); Transcript of Trial Before the Honorable Thomas P. Jackson, Volume 70; liability phase exhibit DX 2718 (attached as Exhibit B) and liability phase exhibit DX 2719 (attached as Exhibit C).

[18] Direct Testimony of James Allchin (liability phase), p. 78; Transcript of Trial Before the Honorable Thomas P. Jackson, Volume 70, p. 93 (Felten Cross).

[19] Transcript of Trial Record Before the Honorable Colleen Kollar-Kotelly, Volume 15, p. 3208-3209.

[20] Issues related to support of earlier versions of Windows are addressed in the discussion of Section 3 of the States' Remedy.

[21] Transcript of Trial Before the Honorable Colleen Kollar-Kotelly, Volume 1, pp. 173-174.

[22] This observation is based upon product reviews that I read during that time (I was a subscriber of Macintosh trade journals at the time).

[23] Direct Testimony of John Borthwick, p. 28.

[24] Direct Testimony of Richard Green, p. 37.

[25] *See* http://perso.wanadoo.fr/levenez/lang/.

[26] "Java Environments," PC Magazine, April 7, 1998, pp. 137-39, 142, 144-46, 150-51, 154-56, 160 (liability phase exhibit DX 2025, attached as Exhibit E).

[27] Direct Testimony of Richard Green, p. 48.

[28] HTML 4.01 Specification, p. 38.

[29] HTML 4.01 Specification, pp. 172-173.

[30] Direct Testimony of Dr. Carl S. Ledbetter, p. 25.

[31] Transcript of Trial Record Before the Honorable Colleen Kollar-Kotelly, Volume 7, pp. 1604-1611.

[32] Direct Testimony of Dr. Carl S. Ledbetter, pp. 26-27.

[33] *See* IETF ("Internet Engineering Task Force") Request for Comments (RFCs) 2518 and 3253.

[34] *See, e.g.*, http://www.openmail.com/cyc/om/00/showfile.cgi?100-1625; http://www.openmail.com/cyc/om/00/index.html; and http://samsungcontact.com/en/developer/contact.php.

[35] Direct Testimony of Dr. Carl S. Ledbetter, p. 33.

[36] Direct Testimony of Dr. Carl S. Ledbetter, pp. 34-39.

[37] Direct Testimony of Dr. Carl S. Ledbetter, p. 46.

[38] Direct Testimony of Michael Tiemann, pp. 71-72.

[39]http://www.zdnet.com/anchordesk/stories/story/0,10738,2595479,00.html.

[40] Sven B. Schreiber, *Undocumented Windows2000 Secrets: A Programmer's Cookbook* (2001).

[41] Prasad Dabek, Sandeep Phadke & Milind Borate, *Undocumented Windows NT* (1999).

[42] Microsoft Press Release, *Microsoft Statement on the Subject of Undocumented APIs* (Aug. 31, 1992).

[43] Direct Testimony of Steven McGeady, pp. 9-11.

[44] Direct Testimony of Steven McGeady, pp. 11-13.

[45]Direct Testimony of Steven McGeady, p. 12.

[46] Direct Testimony of Steven McGeady, pp. 14-15.

[47]iNotes Access for Microsoft Outlook Specification Sheet (*available at* http://www.lotus.com/products/inotes.nsf/allpublic/00D522F4AA9933FF8525685E00572A9D).

[48]Direct Testimony of Steven McGeady, p. 15.

[49] Developer Technical Documentation, *Touring the Pocket PC SDK* (available at http://www.microsoft.com/MOBILE/developer/technicalarticles/toursdk.asp).

[50] Direct Testimony of Larry Pearson, p. 21.

[51] This allegation was also made by Jonathan Schwartz of Sun Microsystems. *See* Direct Testimony of Jonathan Schwartz, p. 31.

[52]Direct Testimony of David Richards, p. 41.

[53] Direct Testimony of Jonathan Schwartz, p. 25.

[54]*See* http://www.iol.ie/~locka/mozilla/mozilla.htm and http://www.iol.ie/~locka/mozilla/plugin.htm.

[55] Direct Testimony of Jonathan Schwartz, p. 25.

[56] *See, e.g.*, Ian S. Graham, *The XHTML 1.0 Language and Design Sourcebook*, http://www.utoronto.ca/ian/books/xhtml1/ .

[57] Direct Testimony of Michael Tiemann, pp. 60-62.

[58] *See, e.g*, Oliver Kaven, "Performance Tests: File Server Throughtput and Response Times", *PC Magazine*, Nov. 13, 2001.

[59] *See* http://de.samba.org/samba/ftp/docs/faq/Samba-meta-FAQ.html#toc1.

[60] Direct Testimony of Michael Tiemann, pp. 62-63.

[61] *See* http://www.freetds.org/faq.html.

[62] Direct Testimony of Michael Tiemann, pp. 63-64.

[63] Direct Testimony of Michael Tiemann, pp. 64-65.

[64]*See* http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnactdir/html/msdn_actdsum.asp.

[65] *See* http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnactdir/html/BuildingADApps.asp.

[66] Direct Testimony of Michael Tiemann, pp. 65-66.

[67] *See* http://msdn.microsoft.com/library/default.asp?url=/library/en-us/act/htm/actml_ref_scpg.asp.