1. The basic technology issues at stake

Colin Jackson*

1. INTRODUCTION

Both sides of the European Microsoft case involved a debate over whether it was appropriate to impose an obligation on a very large player in order to achieve compensating advantages for other players. Thus, with respect to interoperability, the case concerned whether Microsoft should be compelled to deliver details of how to reproduce certain features of its server technology so that other manufacturers could guarantee that their servers would function seamlessly as a part of a Windows network. This involved documenting and sharing technical information to assist in building those features only—and not duplicating the software code involved. As to tying, the case concerned whether Microsoft, having launched a new version of Windows with extra products built into it such that other suppliers of such products might be threatened, could be obliged to redesign Windows to deliver an operating system without those products. In both sides of the case the obligation imposed was balanced against the advantage for others.

From the viewpoint of a participant in the EC case, its technical basis was significantly more complex than its US counterpart. This chapter concentrates on the former. However, a brief comparison with the latter is included later that builds upon the explanations of the technology concerning the EC case.

Many of the articles written about the Microsoft proceedings fail to convey the technical complexity of the issues at stake. The almost byzantine complication of these topics, combined with the unavoidability of dealing with the ferocious, technical details, made it a most remarkable case. The latter conveniently broke down into two parts: interoperability/refusal to supply and tying. It is the former that requires an unusually detailed level of technical understanding, even beyond that normally possessed by

* The views expressed in this chapter are the author's personal views and do not necessarily represent the position of any other party.
a typical employee in an organisation's information technology department. Unfortunately, without an understanding of the technology, it is impossible to follow the case. Furthermore, since the technical concepts are complex, they have tended to be oversimplified or even avoided in the course of the commentary and debate about the legal principles.

Key issues that could be superficially expressed in a few, high-level sentences frequently involved a colossal amount of effort and almost incomprehensible, technical arguments. Examples where such intense debate took place included: the extent to which there were serious problems of interoperability between different software products; defining precisely which protocols had to be released for interoperability; the potential revelation of secrets; the necessary documentation describing how to build software having the crucial features for adequate interoperability; the technical difficulties experienced by SAMBA (represented in the 2006 hearing by intervener, Free Software Foundation Europe); the functioning of media players and the extent to which they were technically linked to operating systems.

This chapter provides the reader with a simplified but comprehensive overview of the technical issues over which much argument took place. It should be noted, at the outset, that these were subsequently elaborated by all the parties, going considerably beyond what is written here. However, no person trying to understand this case properly can avoid getting to grips with these basic, yet hard-to-grasp, technical matters. This includes being able to interpret what was meant in Sun’s original request to Microsoft in 1998 for ‘the complete information required to allow Sun . . . to provide native support for the complete set of Active Directory technologies on Solaris’ and information ‘necessary to provide native support for COM objects on Solaris’. It also includes being able to understand the Remedies proposed in the Decision.

Information technology, like any other technically-based subject, consists of a very wide range of complex minutiae and developments. Such subjects are typically founded on basic facts and hypotheses, on top of which slightly more advanced concepts follow logically. These, in turn, lead to further advances, and so on. The whole edifice can be envisaged as a form of woodland canopy. The basic facts and hypotheses form the tree trunks and everything else follows, in the form of the trees’ main branches leading to sub-branches, continuing on ultimately to a myriad of twigs. Within all this complex structure, to make matters worse, there are interconnections between some of the branches and twigs of the different trees. The purpose of this chapter is to enable the reader to understand the fundamental technology issues involved in the Microsoft case – but not to argue their merits, which is the role of other chapters. This means guiding the reader from the ground to the ends of the particular twigs on which this particular case was fought. This, in turn, implies some inevitable simplification in order to ‘stick to the essentials’. It also means that this is not an attempt to cover the whole of information technology in a few pages. It assumes that the reader has almost no technical knowledge but has, at least, a passing acquaintance with a PC. From time to time, remarks indicate what parts of the case turned on particular technology questions. These references have no special, legal authority and are merely pointers.

2. INTEROPERABILITY

What is an Operating System?

In simple terms, a computer consists of an assembly of hardware modules such as processors, memory, disk-drives and other peripherals. These all intercommunicate via various data-buses. A processor, as a function of its fabrication, will only execute its manufacturer’s predetermined set of instructions. An instruction is an item of data held in memory in the form of zeros and ones that a processor reads and translates as a command to perform an operation, such as, for example, arithmetic, modification of memory contents or jumping to another memory location for the next instruction. Instructions in this form, which a processor can understand immediately, are called machine code. A package of instructions has to be constructed so that the computer can perform some form of useful task. Such packages are known, interchangeably, as software, software programs or just simply programs. These can consist of machine code instructions. They can also take the form of other higher-level instructions, which require translation into machine code. This is explained shortly.

In the early days, programmers not only had to create software that would enable a computer to undertake some form of calculation or data manipulation but they also had to deal with issues such as where to place results in memory in order to initiate printing, or whereabouts on a disk to record data. This quickly became tedious. It eventually became clear that it would be more productive if special, common software could be written that did nothing more than deal with the ‘housekeeping’. It would be separate from the software doing useful tasks. In fact, it would make the running of the computer hardware entirely invisible to the software performing useful work. The special software that performed the ‘housekeeping’ involved in managing a computer’s resources became known as an operating system. Any other software wanting to do something useful would periodically request the operating system to perform a task such
Microsoft on trial

One of an operating system's most important housekeeping tasks was the storing, loading and supervision of an application program.

In the early days of computing, almost all computers were large devices occupying entire rooms. Each computer had an operating system. Jumping ahead to more modern times, computers have significantly reduced in size. Nevertheless, unless they are dedicated to highly-specialized functions, as a general rule, each will still have an operating system.

At first, operating systems were supplied by computer hardware manufacturers, since each vendor's processors had quite different instruction sets. Because the operating system had to deal with the underlying hardware in minute detail, each manufacturer's operating system was 'hand-built' to match. Later, operating systems were programmed more abstractly in high-level languages (explained below), leaving only a small fraction of an operating system to be written in the machine code of the particular processor on which it was running. This enabled some operating systems to be developed that were able to function on quite different hardware.

Different Levels of Software

Although computer processors can only operate by obeying the machine code instructions described above, this is not a particularly convenient form for the large-scale production of software. Instead of having to refer to processors' register contents, memory-location addresses, and the like, and also having to remember what they represent at the time, it is much easier to deal with items that have normal names and where familiar operations are involved, such as 'subtract tax from gross pay to give net pay'. This idea gave birth to so-called high-level languages. Here the software instructions are written in statements that use recognizable names and operations. Special computer programs called compilers translate programs written in such languages into machine code. The completely-translated, machine-code version is then able to be run by a processor. Another approach is to use a special program called an interpreter. This works its way through each high-level statement in turn, presenting the processor with machine code to execute as it goes along.

Programs written in high-level languages are often called source code. Their machine code 'translations' are frequently referred to as binary code. There is a further connotation to the term 'source code'. It can sometimes be taken to mean the confidential, high-level language program that is not publicly available. It may have considerable value.

On the other hand, some source code is made available by developers at no charge, although specific licensing agreements apply concerning its subsequent use or modification. This is known as 'Open Source' software.

Systems versus Applications Software

The simple separation of software between operating systems and applications did not last long. In order to make programmers' use of computers more productive, computer manufacturers provided a series of specialized programs along with their operating systems. These performed basic functions such as copying files, formatting disks and sorting data. These system utilities eventually came to be accepted as forming part of the operating system collection of programs. They were collectively called systems software. They were not referred to as applications.

Operating System Libraries

An operating system enables much software to function by providing shared libraries of commonly used routines, but where some will be used potentially 'on the fly' by programs. This particular type of library is interchangeably called a 'run-time library' or a 'dynamic-link library' (DLL). This consists of a collection of precompiled sub-programs that are loaded into memory dynamically and linked to application programs as required so that they can function.

How Operating System Suppliers Bundle in System Utilities and Applications

The lines between systems software and applications can be blurred by operating system suppliers. This can take place when an application that would normally be supplied by an independent third party is provided instead by the operating system supplier as part of the systems software bundle. In a further step, which links it technically to the operating system, the latter can be made to use the application on a routine basis to provide support for an operating system task. In a step of significantly deeper integration, the application can even be split up and parts of it integrated into the operating system and its libraries. This means that what is left of the application will not work at all unless that particular operating system is present. This is what occurred with Windows Media Player and is explained later. It also took place with Internet Explorer. The consequences of the latter formed part of the US proceedings under US antitrust law.
What is an API?

Applications programs frequently need to request the underlying operating system to supply them with a service. An example of this would be obtaining the time of day and the date. The software interface that is used is called an applications programming interface (API). To enable an API to be used, the operating system’s supplier has to provide thorough documentation to an applications developer. This must describe unambiguously how a particular API is to be used and how it works in terms of what to put in and what will be returned.

This procedure has important implications for a process known as porting. An application may be developed to run on one operating system, use the APIs supplied and become commercially successful. The applications developer may then wish to make the program more widely available by getting it to run on a different operating system (porting it). If this is attempted, the requests to the new operating system via the old APIs will fail, unless the new operating system provides APIs which, as far as the application is concerned, operate identically. To make this possible, documentation called a specification needs to be supplied to the new operating system supplier so that the APIs can be accurately implemented. Without such a specification, the new operating system supplier would be obliged to undertake a slow process of educated guesswork and trial and error, known as reverse-engineering, in order to try and work out how each API works. There is no guarantee that such a process would succeed.

Networked Computers and the Concept of Client and Server

In the 1970s, commercial computing was predominantly undertaken by large, so-called mainframe computers. Desktop access to these was provided by simple terminals that consisted essentially of screens, keyboards and some simple electronics. The greater the number of terminals to be connected, the more powerful the computers had to be. This arrangement had obvious limitations and did not scale well — more terminals added meant that more computing power was needed.

At the end of the 1970s, the personal computer (PC) appeared. At first it was considered to be a novelty or hobbyist device. However, personnel in some companies began to buy them, sometimes clandestinely, to provide an independent means of performing simple computations at their workplace and without interference from the information technology departments that managed the mainframe computers. Although initially ignored, once IBM launched its PC, information technology departments took this to be a seal of approval for the concept. The IBM PC was supported by a substantial marketing campaign and early customers could choose between three different operating systems, of which one was Microsoft’s MS-DOS. IBM’s hardware was also imitated by several other manufacturers. These took licences to load their own version of the hardware with the same Microsoft operating system as IBM used. As a result, such devices were purchased in increasingly large numbers by commercial and government organizations, thus achieving for Microsoft an exceptionally broad base.

Thus, with the advent of PCs, individual computers became scattered throughout organizations, each machine having substantial processing ability. There was also a parallel development in which data networks and communications technologies advanced, enabling these PCs to be attached to high-speed networks. Information technology departments then faced two problems: first, how to connect the PCs to their networks and regain ‘order and control’ over all corporate computing; secondly, how to accomplish this in a way that would not, once again, be limited by the scaling problems described above. The solution turned out to be client/server computing. In this model, one computer, the client, makes a request to another, the server. The server undertakes the operation requested and returns the answer to the client. If, in the process of providing the response, the server needs to make a request to a further computer, it turns itself into a client for that particular operation and makes a further request to another server, and so on. This client/server model ultimately revolutionized computing and enabled very large networks to operate without major bottlenecks building up. The best example of such an arrangement is the World Wide Web over the Internet. Through common usage, desktop computers became colloquially referred to as clients and the computers to which they were connected known in a similarly loose fashion as servers. Communication between such servers was thus frequently referred to as being server-to-server. All this contrasted starkly with the old 1970s computing technology where a large, central computer was connected to a substantial number of ‘dumb’ terminals. It also contrasted with the massive numbers of PCs that became owned by private individuals and were located in their homes.

Some networks connected together different types of computer that ran different operating systems and that sometimes interoperated using different protocols (protocols are explained shortly). Such networks were said to be heterogeneous — for obvious reasons. An example of such would be a network supporting computers running Windows, UNIX and Linux, which are different operating systems (also described below). By contrast,
other networks just connected computers that ran the same operating systems and used the same protocols. These were said to be homogeneous. An example of the latter would be a network connecting an all-Windows environment.

**Operating System Examples and Brief History of Microsoft’s Relevant Products**

For PC-based machines, apart from Microsoft’s products, operating systems that were available at the time included Mac OS X for Apple’s computers and Solaris for Sun’s machines. Solaris was based on UNIX, an operating system originally developed in the 1970s by Bell Laboratories and from which several different versions were subsequently created by various parties. UNIX and another operating system called Linux were designed so that they could be made to work on different computer processors.

Microsoft’s operating system, Windows, was developed primarily for the microprocessors developed by hardware manufacturer, Intel. However, even some later versions of Windows were initially designed to run on DEC’s Alpha processors as well as Intel’s hardware. The relevant history of Microsoft’s operating systems is now briefly explained. These were developed over several years, with improvements being added to successive generations.

Most people will be familiar with Microsoft’s PC client products which, in broad terms, evolved as follows. Its earliest operating system was MS-DOS, which appeared in the early 1980s. This was followed by Windows 3.0 and then 3.1, the Microsoft operating system versions that introduced the now familiar graphical user interface. This was, in turn, followed in chronological order by Windows 95, Windows 98, Windows 98 SE, Windows NT 4.0, Windows 2000 and Windows XP — all successive versions being released between 1995 and 2001. Microsoft also continued its programme of operating system enhancements beyond Windows XP.

The server operating system products will be less familiar. Windows NT was developed to be suitable for both clients and servers, NT standing for New Technology. Prior to this, the Microsoft operating systems were widely felt to be insufficiently reliable to be used as servers in networked computing environments. Windows NT was intended to address this shortcoming. It was released between 1993 and 1996, evolving through several versions from NT 3.1 and culminating in NT 4.0. Its successor server product consisted of three special versions: Windows 2000 Server, Windows 2000 Advanced Server and Windows 2000 Datacenter Server.

Each of these was aimed at different levels of server performance. The successors to these were called Windows 2003 Server and originally consisted of four versions: Standard edition, Enterprise edition, Datacenter edition and Web edition. Microsoft also continued its release of improved server operating systems beyond these.

**Distributed Systems and Transparency**

In the preceding explanations, it has been implied that a program is an item of software that runs on a single computer, with help being provided by the underlying operating system. However, in a networked environment this could mean that some resources were underutilised whilst others, running at maximum capacity, acted as bottlenecks.

To overcome this, some programs can be designed in the form of components such that some of the latter may run on different computers on a network. To shield programmers from having to deal with the potentially ferocious complexity of managing all this, special systems software has to be provided. This makes the running of such distributed systems invisible to the programmer. This transparent distribution of resources across a network is an important facet of the Microsoft case.

**What are Protocols?**

It is, therefore, evident that, in a networked environment, different software components or even entire programs will sometimes need to communicate with each other, often across the network. To do this, they interchange packets of data in a structured way according to a predetermined set of rules. These rules are known as protocols. Protocols define the format, sequence, timing and error checking involved in these exchanges. Very many different protocols exist, each having been designed to fulfil a specific purpose.

For example, one could be used simply to convey basic packets of data over a wireless network. Another could operate at a much higher level and be used to exchange an e-mail. In this way, once a basic data transportation mechanism had been established with one protocol, another could take advantage of this and provide a higher-level service. Thus protocols can run within protocols. Some protocols form a coherent layer of services, the top one requiring the presence of the protocol immediately underneath, and so on. This is unsurprisingly known as a protocol ‘stack’.

The concept of protocols is fundamental to a proper understanding of the Microsoft Decision and its Remedy.
Distribution, Object-oriented Programming and the (Brief) Role of Microsoft's COM/DCOM

To proceed further, it is necessary to understand a little about a particular programming technique known as object-oriented programming. The term 'object' was used extensively throughout the Commission’s Decision, and ensuing arguments, and related to this concept.

Originally, programs were written as a set of programming steps that precisely worked through the computation to be undertaken. The individual, computational steps could also, as appropriate, perform operations on data. The data was accessible to the entire program and was stored separately from it. High-level languages such as Basic or C are examples of this so-called procedural approach. Within such a sequence of steps, a program could call, whenever required, for a common procedure (sometimes called a subroutine) to be performed. It is only a small step to realize that these procedures could also be distributed around a computing network, rather than remaining on one computer. Specialized systems software was developed that would enable a program to call a procedure that could, potentially, be executed on a remote computer but which, most importantly, shielded the complexity involved from the programmer. This was termed a remote procedure call or RPC.

Object-oriented programming is different from the above procedural form. Its ultimate aim was to permit the creation of reusable software components. The idea behind such an approach is as follows. A so-called software object combines in a single entity both processing (these operations are more formally known as methods) and data (known formally as attributes). As stated above, in traditional programming approaches these are normally kept quite separate. In short, an object encapsulates a data structure, the data it contains and the code that operates on it in such a way that its inner workings and data layout are hidden from the outside world. If the face that it presents externally remains unchanged, then its internal efficiency can be safely improved without compromising any software that uses it. These characteristics alone make software more manageable. Software objects having the same common operations and attributes are bundled together in what are known as classes. In this way, any object which is declared to be a member of a particular class automatically inherits the common operations and attributes of that class. When programs are run, whole structures of objects are created (and destroyed) dynamically. Each individual executing structure is known as an instance. By using these concepts, a program can be constructed as a collection of cooperating objects. Smalltalk and Java are object-oriented languages.

Once again, objects can also be distributed around a computing network, as opposed to remaining on one computer. The communication between objects, no matter where they are, corresponds to one object making a request (more formally called invoking a method) to execute an operation on another. Again, specialized systems software has to provide this service for the objects. In this way, with the appropriate network and systems-software infrastructure in the background, an application programmer could be entirely unaware that a computer on the other side of the world was performing part of the work.

In practice, it turned out to be difficult to achieve this in a world of assorted computers and operating systems. Getting different manufacturers’ operating systems, libraries of standard software routines and network protocols to interwork efficiently was a daunting task. An early attempt to provide a public standard for such software, that would be independent of operating systems, was called CORBA (Common Object Request Broker Architecture) developed by the Object Management Group, an industry-wide standards organization. With CORBA, objects written in different languages and distributed around a network could work together as though they formed a single program on a single computer. To make this possible, CORBA required software objects to ‘publish’ the interfaces that they presented to the object-managing software system. This was done by means of what was called an Interface Definition Language (IDL).

Microsoft developed its own proprietary, distributed, object-management system that used IDL to specify the interfaces. This specialized system software was called COM/DCOM. COM stands for component object model and DCOM for distributed component object model. DCOM extended COM and enabled objects to be located on different computers within a network. A further development, COM+, expanded the services offered. Interestingly, as part of its internal functioning, and hidden from the programmer, this Microsoft software made use of RPCs to support the complex, inter-system linkages required. Microsoft’s RPC was based on that of the OSF (Open Software Foundation) DCE/RPC (distributed computing environment/remote procedure call).

This Microsoft RPC facility was, of course, available to be used by programmers writing software in the traditional way. Furthermore, it is important to underline that any program, no matter how it is constructed, may wish to request the operating system to perform some specific service. To do this, the software code simply makes the request by using the appropriate API. In reality, unbeknown to the programmer, this might involve procedures resident on remote computers.
The COM/DCOM facility, RPCs and IDL play an important part in the server-to-server protocols that lie at the heart of the documentation remedy in the Microsoft case. COM information was specifically requested in Sun's 1998 letter.

Standards and Documentation

Many protocols are used by a great number of different computer systems across the world. Self-evidently, this implies that many of the protocols have had to be agreed in advance by accepted industry bodies. Furthermore, these protocols have had to be clearly and unambiguously documented so that different parties in different locations could build computer systems that used them correctly. Examples of two protocols pre-agreed by an industry body are Internet Protocol Version 4 (IPv4) and Transport Control Protocol (TCP), both ratified by the Internet Engineering Task Force (IETF). Most people will at some time have come across the term TCP/IP, reflecting the combined use of these two protocols by communications networks, including the Internet.

In contrast to such public protocols, some have been developed privately by operating system suppliers, for example. In either case, where an operating system has become very widely used, these protocols can become a de facto standard. If such a protocol is proprietary, it then falls on the supplier to produce adequate and accurate documentation for its users. An example of a proprietary de facto standard in the past was Systems Network Architecture (SNA), a collection of networking protocols, which was originally developed by IBM in the early 1970s.

Software developers who use protocols created by others rely almost entirely on the quality of the documentation made available to them in order that their implementations work correctly in all circumstances. If a protocol turns out to operate in an unexpected manner in some circumstances, this needs to be recorded too. Such idiosyncrasies can be very important. Thus, in a similar way to APIs, programmers need complete, clear and unambiguous specifications of protocols in order to be able to implement them accurately.

Documentation may also include what is called a reference implementation. This is an example of the code that could be written for the implementation of a protocol. It represents a fictitious example and is one of the potentially infinite ways in which such computer instructions could be expressed. It helps programmers understand what the documentation intends that they do. It does not reveal any party's actual source code.

Extensions to Standard Protocols

Standardized protocols developed by accepted public organizations often incorporate places where some extra information can be added and interchanged between parties. The extra information may be well-defined or it may be unspecified, it being left to an implementer to add a desired extra feature. Sometimes a standard protocol may be unofficially modified for proprietary use. In either case, such modifications are usually referred to as protocol extensions.

What are Work Group Servers?

Work group servers are inexpensive computers that provide basic services to office workers' client PCs in the form of the sharing of printers, files and other resources, and the administration of the access to these or other services by either users or groups of users linked to the work group network. Such work groups are frequently made up of small to medium-sized networks. Many office workers will be familiar with such servers, which are found scattered around their places of work. They will also be familiar with the ability to be able to share files with other users on the same network. In the Windows environment, this file-sharing involves two services called CIFS and Dfs. This is explained below.

Work group servers frequently run other small, shared applications as well.

The Microsoft case focuses on work group server operating systems.

File Services

As stated above, one of the core services provided by work group server operating systems is the provision of file services.

So-called network file systems allow users to share files over a network. An important feature of such services is that they hide from users the complexity relating to where individual files are stored. The approach is to make it appear to users that the files that they are accessing are located on a locally-attached disk, whereas, in reality, they reside on different servers spread across the network. This provides a logical view of the files rather than an exact physical view. Microsoft's basic system was called Common Internet File System (CIFS). The protocol used was also sometimes referred to as CIFS. This was developed out of an earlier protocol called Server Message Block (SMB), originally created by IBM in the early 1980s. In fact, apart from a few small changes, CIFS and SMB are basically the same protocol. It is thus common to see references to CIFS/SMB. The
word 'Internet' in CIFS originates from Microsoft’s application to the IETF in 1996/1997. The application time expired but the term 'Internet' remained in the name.

Microsoft’s Distributed File System (Dfs), which used CIFS, was a further ‘add-on’ development in the late 1990s for Windows NT 4.0 servers and clients. Enhancements continued with later versions of Windows operating systems. As stated above, with CIFS/SMB, a user can access a file on another computer that has been declared to be ‘shared’. The file appears to the user as though it is resident on that user’s hard disk. With Dfs, an administrator takes a group of files that are located on different servers and, with this facility, enables a user to view them as a set of files contained within a newly-named common folder that also appears to be resident on a user’s hard disk. In this way, the complexities of accessing various files on different remote servers are completely hidden from users. The purpose of this service is to simplify the sharing of groups of distributed files. Users are spared having to perform laborious searches using ‘My Network Places’ on their Windows-based computers.

The adequacy of documentation available for the Dfs protocols formed part of the Microsoft case.

Security and User and Group Administration Services

An important element of the services provided by a work group server operating system is ensuring that access to the resources available is controlled in a secure fashion. This means that single or groups of users must only be able to use the services for which they have been given permission. There are two aspects to the achievement of this: authentication and authorization. The operating system has to handle both. Authentication is the act of verifying that a user really is who he or she purports to be. Authorization is the process of checking just precisely what a user is permitted to do and whether providing or blocking access accordingly.

The Microsoft case involves some of the specific details of how this was achieved in practice. However, before proceeding with the explanation, it is necessary to understand the notion of directories.

What is a Directory?

It takes very little imagination to realize that, in a network of clients and servers, resources such as printers, disk drives, scanners and the like could be spread out amongst the various computers in a very complicated way. Furthermore, the information concerning the user and group administration services relating to all this would need to be stored and then either

accessed or updated by the various computers concerned. This conveniently leads to the concept of what is known as a directory.

This is effectively a collection of information concerning all the resources contained within the network. This even covers program files, including those connected with operating systems. It also includes user and group administration services information. A directory can be thought of as a form of ‘Yellow Pages’ and ‘White Pages’. In the former, resources can be located according to their properties. In the latter, they can be looked up by name. Each item in a directory can be envisaged as having an account that contains a description of all its properties, including its name. The concept of directories is used in many contexts and is not just limited to work group server operating systems. However, the concept of directories was particularly important in the Microsoft case.

In a collection of networked computers, the directory resides on a designated server or even servers.

Over the years, a whole specialized area of technology has grown up around directories. This is concerned with issues such as how to store and retrieve the information efficiently and how to update a directory when the networks of computers involved are very large and can even be spread across different time zones in different parts of the world. Directories became so important that international standards (called ‘X.500’) were even developed in the 1980s in the hope that they would be widely used. These standards related to subjects such as common ways of accessing directory data and the organization of the latter. They also defined a protocol for accessing a directory. Given the information technology world’s penchant for using acronyms, this was, unsurprisingly, called DAP (Directory Access Protocol).

What is LDAP and What was it Intended to be Used For?

When DAP was first designed, it was assumed that the exchange of data over computer networks would take place in a particular way, known as the ‘OSI model’. DAP was designed to be able to operate in this environment and dealt with a lot of the complexity that was involved in the communications model envisaged, such as connection set-up. However, the advent of the Internet and the protocols that were employed there meant that data communications evolved in a simpler way than the full ‘OSI model’. This meant that a lot of the complexity involved in writing programs that used DAP became an unnecessary overhead, as well as the substantial computing load imposed on the processors then available. As a result, a simpler tool than DAP was developed that would operate directly over the Internet protocols involved and that omitted the unwanted or
rarely-used complications. This became known as LDAP (Lightweight Directory Access Protocol). It also eliminated a lot of the unnecessary DAP complications such as its complex way of referring to the names of directory items. Although simpler to use, LDAP is not a universal panacea. It is effective where rapid directory queries are involved or infrequent directory updates are required. It is thus suitable for looking up a user's password but not for a bulk update of a large amount of directory data.

It is important to note that programs can be designed in very many ways and they can simply use LDAP to communicate with the server or servers that hold the directory information. As such, LDAP communication may be thought of as a kind of 'language'. Directory servers can be designed to 'talk' only LDAP, for example. However, and this is important, they can also be designed to use other, different means to receive and send data, particularly when large amounts are involved. In such cases, LDAP may be just an added-on form of directory communication.

This point is also important for the understanding of the Microsoft case.

Role of Domains in the Microsoft World plus Domain Controllers

Although the idea of having a directory of all the resources, hardware and software making up a networked environment can seem to be logically appealing, it can be readily imagined that administering a gigantic register of everything could very quickly become a nightmare. For this reason, in a Windows environment, collections of client and server computers that share user account and security information can be broken down into separate groups called domains. It should be noted that, as a matter of principle, this is a logical grouping for administrative purposes. It is not a physical one. In other words, individual members of a domain could be scattered around different locations. Nevertheless, a member of a particular domain can 'log on' to it and, depending on the various authorizations stored, have access to the different resources also registered in that domain. A domain is therefore a key administrative concept and forms an integral part of the user and group administration services described earlier. The person who supervises and controls the workings of a particular domain is known as its administrator. All Windows operating systems from NT 3.1 to later versions use domains.

In its simplest form, a domain could be used to include all the users for a single company. However, in large organizations, it could, for example, be more convenient to have different domains for each country, state or even city location. Furthermore, if one company purchased another, given the ubiquity of Microsoft operating systems, it would be more than likely that the new company had already implemented its own particular domain structure. In a further twist, personnel could be relocated to different geographical locations but still remain members of their original domains. In short, an organization could have several domains, each containing members that were spread around different geographical locations, with each user being limited to his or her domain. It is self-evident that in such circumstances, users in one domain might need to use or access resources contained in others.

To overcome the problem of users being locked into and limited to their own domain, the concept of trust relationships was eventually implemented by Microsoft. A trust relationship is a form of administration link between two domains that concerns authentication. One domain is declared to be the trusting domain and the other the trusted domain. The trusting domain simply trusts the other to accurately authenticate the trusted domain's own users. Note that it does not allow the trusted domain to authorize anything on its behalf. Trust is a one-way, authentication relationship. The trusted domain does not accord the trusting domain the same privilege. For this to happen, a second relationship has to be established the other way around. Such a mutually reciprocal arrangement is, unsurprisingly, called a two-way trust relationship. It is important to note that trusting another domain does not give the trusted domain's users the right to access the trusting domain's resources. For this to happen, the trusting domain still has to assign authorization permissions to individuals in the other domain in exactly the same way as it would for its own members.

There is a further complication to the concept of trust relationships and this is the notion of a so-called transitive relationship. Suppose that domain A establishes a two-way trust with domain B. Suppose also that domain B establishes a separate, two-way trust relationship with domain C. If the trust relationships are transitive, the trust relationship granted by A to B is automatically extended to C, i.e. A also has a two-way trust relationship with C. One-way, two-way and transitive trusts became possible with Windows. The significance of this is explained below.

This notion of trust relationships between domains leads to hierarchies and other linkages of trust relationships that eventually became features of Windows. This gave rise, first of all, to the concept of 'trees'. A tree is a hierarchy of domains linked by transitive, two-way trust relationships. This can be thought of as a form of family tree. At the top of the tree is the 'parent', called a root domain. Immediately under that are its 'children'. Immediately under each 'child' are the 'grandchildren', and so on. Of course, although the root domain will be there, the tree underneath is not obliged to be symmetrical. Some 'children' will not have any
The basic technology issues at stake

Just exactly how domain controllers work and manage the functioning of a directory depends on the version of the Windows operating system that is being used. This is explained later. However, the role and functioning of domains and domain controllers are crucial concepts for the understanding of the Microsoft case. By way of example, a great deal of the technical argument in the 2006 hearing required a detailed knowledge of these concepts.

Work Group Servers and Heterogeneous Networks

Within a corporate environment, computer services can be applied to run several different parts of a business, e.g. manufacturing systems, accounting systems and product design systems. In addition, computer services are also provided for general administrative tasks at the desktop and include file, print and other services. All of these systems may use the same corporate network, although they may have very limited interoperability. For example, the manufacturing system might make certain status reports available by publishing them on an internal web-server. Permitted desktops would then view such content via their browsers. The network is said to be heterogeneous because it is supporting all the intercommunications between the disparate systems. In this sense, heterogeneous networks are very common.

A sub-set of the above is the provision of computer services for general administrative tasks. As explained earlier, this is known as work group computing. Within this sub-set, the problem is to connect seamlessly at all levels the servers that support the administrative services delivered to the desktop. The case describes the obstacles that stood in the way of achieving this. Taking the work group sub-set of corporate computer services, a heterogeneous network of servers within the same domain became increasingly difficult to achieve with the introduction of a Microsoft product called Active Directory. The reason for this is quite complex and is explained in detail in the sections that follow. As such, heterogeneity became reduced in this sub-set of corporate computer services. Various proposals were put forward that only achieved limited interoperability similar to that described in the preceding paragraph. This was insufficient to provide the interoperability required between work group servers in the same domain. In that sense, heterogeneous networks became relatively rare.

Microsoft's Directory Products Leading to Active Directory

Domains are clearly a particular concern of Microsoft's server operating systems. Microsoft originally introduced the concept of domains
with Windows NT. However, at this point, domains were effectively self-contained, unrelated entities, although trust relationships could be set up between them. Trees and forests did not exist. With Windows NT, each domain's management was controlled by a server called a 'primary domain controller'. It was given this name because, in the event of it failing, another server, designated as a 'backup domain controller', could take over. There could be more than one backup domain controller or even none. The management of a domain involved the primary domain controller managing a master collection of records called a security accounts manager registry. With the usual propensity for acronyms, this was known as the SAM registry. Interestingly, this did not support the X.500 directory standards or LDAP. Data had to be accessed via a special Microsoft API. The primary domain controller periodically sent copies of its SAM registry to the backup domain controller(s). Apart from being able to take over if the primary controller failed, the backup domain controllers could also share the workload in a busy network environment. Servers not involved with any of these activities were simply known as member servers of a domain, their resources being recorded in the SAM registry. As mentioned earlier, a person called an administrator had to set up and maintain the primary and backup domain controllers.

With the launch of Windows 2000, a radically different approach was taken to the organization and the management of domains with the advent of a significantly improved service named 'Active Directory'. The reader will recall that Sun's letter requested information concerning Active Directory technologies.

In brief, Active Directory introduced the concept of trees and forests, allowed the use of LDAP, employed different security techniques and allowed many more users to be managed. Active Directory also introduced a new user and group administration service named Group Policy. This applied a standard set of security options and controls to users designated to be part of a particular group. Group Policy thus simplified the administration of users and computers. Thus when a user logged in, if the Group Policy feature was enabled, Active Directory would check which policy applied and then provide the requisite desktop configuration and permissions for that user. One of Active Directory's most important changes, however, was the totally different way that domain controllers worked. Here, it used a technique called multi-master replication. This is another key concept in understanding the Microsoft case and is explained later.

First, though, it is necessary to understand the security technique changes as these, too, are crucial to following the case.

Security Within the Operating System Environment and the Role of Kerberos in the Microsoft Environment

With Windows NT, the authentication protocol that was used was called NTLM. This was proprietary to Microsoft. It was basically used in the processes between a PC client, a domain controller and a server where authentication of the client was involved. This ultimately allowed a server to provide the authorized services for a particular user, the details of which had been stored in the SAM registry in the domain controller.

With Active Directory, there was a major change made in the protocol used. Authentication was based on a protocol called Kerberos. For various technical reasons, this was a significant improvement on NTLM. Kerberos, though, had not been developed by Microsoft but by MIT. The specification for this protocol was in the public domain (it was specifically so-called version 5 and documented by the IETF, mentioned earlier). However, Microsoft took this public standard but added a proprietary extension to it. To understand the significance of this, it is necessary to have at least a basic understanding of how Kerberos works in the authentication process.

This operates through the issuing and exchange of so-called tickets. Starting at the beginning, a user's password is stored on an Active Directory domain controller in an encrypted form. When a user logs on at his or her PC client, a process familiar to most people, an encoded version of the password is passed to the domain controller. This compares the user's password to the stored version. If it matches, the user is deemed to be authenticated. At this point a so-called 'ticket' is generated. This contains the name of the Windows user and the time that the ticket is to remain valid. A good analogy is an entrance ticket for a country fair. This ticket is sent to the PC client. However, this ticket is special and can be thought of as a master ticket or a 'ticket-granting ticket' in that it is going to be used to generate further tickets. When the user's PC client needs to access particular resources on another server, it sends this master ticket to a domain controller. This transmitted ticket consists of two parts: one that is in clear text and the other that is protected by strong encryption. The domain controller then generates a second so-called service ticket which only remains valid for a given time period and for that particular purpose. A good analogy for a service ticket is a voucher for an individual ride at a country fair. This service ticket, which in Microsoft's version also contains authorization data (see shortly below), is presented to the server that will provide the service and access to the required resource is then granted. Provided that the user remains logged on, tickets that expire are renewed.
The Kerberos system requires that tickets are issued by a software function called a key distribution centre (KDC). In Active Directory, this KDC function is incorporated in every domain controller.

The objective of Kerberos is to provide a secure authentication mechanism. However, in the tickets, the IETF standard does allow some authorization data to be carried in a specially-defined area of the Kerberos protocol called the 'authorization data' field. Microsoft uses this to carry a proprietary structure called a privilege access certificate (PAC, a further acronym and sometimes referred to as a privilege attribute certificate). This is carried within the encrypted part of the Kerberos ticket. Microsoft uses this PAC to include authorization data in each Kerberos ticket-granting ticket and ticket. Furthermore, the encrypted part of a ticket also contains an additional, security-check item which is, itself, further encrypted. It is important to underline that the content and structure of this PAC information is specific to Microsoft. It is not defined in the IETF standard, apart from the fact that the 'authorization data' field is available for use.

Thus, in summary, in the Kerberos process, a KDC will initially generate a ticket that includes Microsoft-specific authorization information carried as a PAC in the 'authorization data' area. This PAC will be inherited and carried along by subsequent ticket requests or renewals. There is a fundamental consequence of this. Any external party managing to observe a totally-decrypted Kerberos transaction generated by Active Directory would be able to recognize the information being transferred because it would correspond to the standard format specified by the IETF. However, they would not be able to interpret the information contained in the PAC. This is also crucial to understanding the Microsoft case.

A corollary is that any Windows 2000 computer within a domain, including a work group server, would reject tickets that contained PACs that did not comply with Microsoft's standard. This is also a crucial issue as it fundamentally affects the acceptance of a non-Microsoft server within a group of Microsoft servers.

How Does This Affect LDAP?

The following short section forms a particularly dense 'acronym soup'. However, it has particular relevance because it provides the background as to why some of the technical solutions proposed during the course of the case would or would not work. It also explains the relevance of Sun's allegation that Microsoft had made undocumented extensions to something called 'SASL'.

The Windows 2000 generation and later operating systems in this case used the international standard LDAP version 3. This was designed to use as an option an authentication framework called Simple Authentication and Security Layer (SASL). The idea behind its use is as follows. Instead of an LDAP information exchange just taking place in the open over the basic communications protocols, TCP/IP, referred to earlier, an authentication protocol is first 'negotiated' by SASL and then a security protocol is inserted between LDAP and TCP/IP. This means that, following authentication, the subsequent LDAP information exchanges will be carried securely across the network connections.

To make this clearer, SASL is a special authentication framework that first finds out what secure authentication protocols are available, secondly performs the necessary work to 'fire up' a selected authentication protocol plus subsequent security protocol, and then finally drops out, letting the now-secured communication continue. In order to do this SASL uses a series of 'mechanisms' that consist of the patterns of information exchanges that are required to initiate a whole series of authentication protocol exchanges. Thus, SASL mechanisms are defined for several authentication exchanges. One of these mechanisms 'fires up' Kerberos version 5, explained above. When a PC client that wishes to make a series of LDAP transactions successfully initiates a SASL process that will start up the various exchanges involved in authentication, this is technically known as a 'SASL binding operation'.

As a function of the selected protocol, various messages will then be interchanged between the PC client and the relevant domain controller. In the case of Active Directory, Kerberos version 5 is selected and is initially used by LDAP.

It should be noted that the Kerberos interchanges conveyed contain the proprietary PAC, described above, although this is independent of SASL. If a secure authentication protocol exchange such as Kerberos fails, a PC client can make a so-called 'anonymous' LDAP request. An 'anonymous' authentication is a standard mechanism used by LDAP but provides extremely limited, read-only access to directory resources.

Part of the dispute concerned as to whether or not the SASL binding operation had also had proprietary extensions added to it by Microsoft. It also concerned the extent to which LDAP could be used to access and update Active Directory information.

To summarize with respect to both of these issues, when LDAP is used it 'fires up' SASL. Sun alleged that Microsoft had made undocumented extensions to SASL. In any case, when either SASL or Kerberos (which it initiates) fails, LDAP is reduced to being able to provide only very limited information.

This results in a significant handicap to a non-Microsoft server.
Key Distributed Feature of Active Directory’s Functioning

In contrast to the workings of the directory service available in Windows NT, Active Directory uses a technique called multi-master replication. The essential elements of this will now be explained.

In Windows NT, each domain has a primary domain controller and, potentially, other backup domain controllers. Self-evidently, the primary domain controller is 'in charge', unless it fails. From time to time, the master information on the primary device is copied to the backups. With Active Directory, the situation is different. All the domain controllers within a domain are peers. With such a system, if any change is made to one of these domain controllers, the modification is propagated to the others through a special process. This means that there is no master copy of a directory. The name given to the propagation process is multi-master replication. The process for propagating changes in Active Directory is very efficient. It only passes on data changes and, even then, the net result of several changes to the same object. Although this may sound simple, the process is quite complex. An example of the latter is how to handle the accessing of a directory object on one domain controller, where the object has been updated but the update propagation has not yet been received. It has to be remembered that with the concept of forests and trees, the span of such interchanges could be very large and even involve multiple domains spread over different continents. The entire replication process takes place using special, Microsoft-specific RPCs and even a standard e-mail protocol.

With Active Directory, it is necessary to create and maintain a topology of physical network connections to enable the domain controllers to intercommunicate efficiently during replication. For this, Active Directory uses what are termed 'sites'. A site is a physical network covering a particular location that has good physical communication between all parts of that network. It is usually a local area network, the 'LAN' with which most people are familiar. The topology of connections within (and between) sites is independent of the forest/tree domains structure. An administrator also has to set up the topology of these interconnections. This ultimately determines how domain controllers’ replication takes place. This is worked out in the following way. Each domain controller regularly runs a software component called a knowledge consistency checker (KCC). This creates the replication schedule and the links to be used within a site for domain controllers to propagate their replication data between each other. The KCC can dynamically adjust these every time it runs. If a new domain controller is added somewhere, the KCC automatically amends the links and replication schedule accordingly. Sites may also have to have connections between them. These inter-site connections are also set up by an administrator. Once again the KCC works out the best way to use all the available connections, including taking account of unavailable connections. A domain controller's KCC normally runs every 15 minutes.

It can thus be seen that Active Directory consists of many domain controllers, none of which is the master, potentially spread over a large geographic area and all periodically propagating directory-change information to each other as appropriate. It thus forms a genuinely distributed system.

The Active Directory protocols used for transactions within a domain are considerably more varied than those relating to exchanges between domains. That is a consequence of the replication traffic.

The distinction between intra- and inter-domain interoperability played an important part in the Microsoft case. In the April 2006 hearing, Microsoft presented a boundary to what it believed it should disclose and termed this 'off limits' environment the 'Blue Bubble'. This appeared to include the intra-domain and inter-site protocols, amongst others.

Upgrade Migration Path for Microsoft Operating Systems Leading to Active Directory Native Mode and the Technical Consequences

When all the domain controllers within a domain are running the Windows 2000 (or later) operating systems and the full functionality of Active Directory is available to users, Active Directory is said to be operating in 'native' mode. In other words, the system is operating fully without constraint and users are able to make use of the features that it was designed to provide.

However, it can be readily imagined that, following its introduction, as new computers running the Windows 2000 operating system were introduced to networks or as operating systems were simply upgraded on existing computers, some domain controllers would still be running Windows NT whilst others would be using Windows 2000. To allow for this, Active Directory was able to operate in what was termed 'mixed mode'. This allowed domain controllers using Windows NT to coexist in a domain with those running Windows 2000. This mixed-mode operation was the default setting for Active Directory. Its purpose was to provide a gradual way of migrating a domain from the old Windows NT directory system to Active Directory, provided with Windows 2000, without requiring an immediate, all-or-nothing conversion. So that Active Directory could provide services to servers in a domain that was running the old, Windows NT operating system, a single, Active Directory domain controller had to be nominated to act as the primary domain controller that the old system required.
Mixed mode, however, came at a price. The full services of Active Directory (in particular Group Policy, described earlier) were not available. To be able to take advantage of the complete range, native mode had to be selected for a domain. This required that all its domain controllers (including the old backup domain controllers) had to be upgraded to Windows 2000. Furthermore, conversion to native mode was a one-way process.

A corollary of this is that conversion to native mode in a domain locked out the addition of any potential, additional or replacement domain controllers for that domain if they ran the old Windows NT operating system. This is also fundamental to the understanding of the Microsoft case.

Sometimes, organizations that were running native mode Active Directory acquired new companies that were still running the old NT systems. There was thus a requirement to link an NT domain to a native mode system. This was done through the concept of defining a so-called external-trust relationship. It had, however, a limitation. An external-trust relationship had to be specifically set up between two named domains, the NT newcomer domain and a named domain within the native mode system. However, an external trust is two-way but not transitive. That meant that the NT domain was only trusted by the named native-mode domain. The trust relationship was not propagated throughout the native-mode forest. For other domains in the latter to share trust with the new NT domain, separate, parallel, external-trust relationships had to be established with each of them on an individual basis.

SAMBA’s Problem

Now that the consequences of a Windows NT domain controller faced with Windows 2000 and Active Directory have been understood, this is the point at which to introduce the intervening party, SAMBA. It confronted significant challenges.

SAMBA is the name given to a collection of software programs that enable a server using a non-Microsoft operating system to provide file and print services to Windows PC clients by means of the CIFS/SMB protocol. The development of this began informally in 1992 when an Australian, Andrew Tridgell, succeeded in connecting a computer running a Unix operating system to another in his home running a Microsoft operating system and which made the Unix-running computer appear as if it were a file server to the Microsoft machine. He made this possible by reverse-engineering the SMB protocol. Since then, the SAMBA group and its software have grown. SAMBA runs on other operating systems, including Linux. The software is an Open Source product and is licensed freely under the terms of the GNU General Public Licence. Participants in the SAMBA project provide their services at no charge to the group.

SAMBA ‘advanced’ from the SMB protocol to implementing the CIFS/SMB protocol.

At the time of the case, by using SAMBA, a server running a UNIX-based operating system could provide file and print services, including Dfs interoperability, and was also even able to act as a primary domain controller in a Windows NT environment. It could not act as a backup domain controller in the latter.

With the advent of Active Directory with the Windows 2000 operating system, as explained earlier, networks of computers running both systems had to coexist. In particular, some domain controllers would be running Windows NT whilst others used Windows 2000. In this situation, Active Directory ran in its default mixed mode. Here, in a domain running the old, NT operating system, a single Active Directory domain controller had to be nominated to act as the primary domain controller, which the NT system required. However, a server running SAMBA was not able to act in this Active Directory role. As it was also not able to act as a backup domain controller, the jump to Active Directory with Windows 2000 meant that SAMBA’s ability to be a domain controller had ceased. However, a SAMBA server could still be a member server in a domain.

SAMBA’s situation was described in the Commission’s Decision. It also provided a key battleground for the interoperability argument that took place at the 2006 CFI hearing.

Problem of a Non-Microsoft Server/Operating System Becoming Part of a Group of Microsoft Servers/Operating Systems

If a server running a Microsoft operating system is added to an existing Windows environment, as described above, it will efficiently interoperate with all the existing computers: PC clients, domain controllers and member servers. If a server running a competing operating system is added, it will have to perform equally well; otherwise users will ultimately perceive that the expected, ‘behind-the-scenes’, technical transparency no longer exists. This could create unacceptable overheads, inconveniences, or even failure. An example of inconvenience could be a systems administrator having to create a new domain and a user having to log on twice, once to access the normal Microsoft available resources and again for the resources of the non-Microsoft server in that domain. During the case, there was debate over whether such inconveniences had a significant effect on customer behaviour or purchase decisions.

A first requirement to enable technical transparency to be maintained
is that the non-Microsoft server operating system has to interoperate in exactly the same way as a Microsoft server. This means it must be able to use all the relevant protocols and APIs in a conformant manner. This, in turn, means the new operating system’s supplier would need to have access to complete and accurate documentation concerning the use of the requisite protocols as well as the necessary APIs. The latter is necessary because Microsoft computers will be trying to access the new server, expecting it also to be a Windows-based device.

With Active Directory, there is also a second and deeper problem and this concerns domain controllers. It was explained earlier that Active Directory’s domain controllers are all peers. In fact, they are all operating independently, each deciding what directory synchronizing data needs to be interchanged with others and each running their own KCCs at regular intervals. The Active Directory model is a distributed system that requires each independent domain controller to operate according to identical rules and to come to identical decisions in identical circumstances. This means that a form of group behaviour is involved.

A good analogy to this situation is a hive of bees. Each bee has a preprogrammed behaviour where intercommunication between the bees is such that an independent movement by one insect, with its body for example, is immediately understood and potentially acted upon by others. In this way a whole hive intercommunicates and ultimately behaves as a corporate body. However, if an artificial bee is introduced that has an incomplete or faulty set of movements, then its seemingly innocent scratching of a leg might, for example, be propagated throughout the colony and inadvertently send the entire hive off to a non-existent flower grove. Similarly, if a non-Microsoft domain controller did not make exactly the same assumptions, perform exactly the same replication calculations and decisions, and behave as its interconnecting Microsoft peers, the propagation of replication information across the correct links could be fatally compromised. This could lead to chaos and an irretrievably broken directory system. This means that a really complete description and explanation for the use of protocols and APIs, and which covered any detailed implicit assumptions, would have to be available and be faithfully followed by the developer of the non-Microsoft operating system.

It is important to note that the last sentence does not mean that a non-Microsoft domain controller would have to behave in an identical fashion for every other conceivable activity that it could undertake, for example the way that it stored its files.

It should now be evident that just providing any communications protocol would not solve this particular problem, it would have to include the directory replication protocols.

Problem of Documenting the Interfaces to Permit Interworking Without Revealing Secrets Concerning How the Rest of Windows Works

Because of the absolute requirement to include any implicit assumptions made in producing the necessarily complete documentation required, it might be argued that revealing some of these assumptions would automatically mean revealing confidential information that concerned the inner workings of the Windows operating system.

Returning to the bee analogy, suppose that two bees each discovered two flower gardens that were each a major, new source of nectar but where both gardens were equally attractive. If one bee decided that one garden was to be exploited first while the other made the opposite decision, both bees would enter the hive, make their requisite ‘body movements’ and end up by circulating conflicting information to the rest of the hive. Suppose, however, that bees had been preprogrammed through evolution to always break the tie in exactly the same way and that the tie-breaking mechanism involved using the altitude of the flower gardens: in case of a tie the lowest altitude is always the winner. This way, both bees would return to the hive, exhibit the same ‘body movement’ behaviour and harmony would be preserved.

In a complete description of each bee’s communications behaviour, it would, however, be necessary to reveal the altitude tie-breaking mechanism. Immediately, an observer would know that part of a bee’s internal workings involved altitude measurement.

The above analogy refers to the Active Directory mechanism used to propagate directory change information. Domain controllers do this by using identical decision-making mechanisms, even in tie-breaking situations, although they operate independently. This produces the cooperative group behaviour that successfully propagates the changes without making errors or creating inconsistencies. Microsoft claimed that if they revealed the details of these protocols, they would reveal hidden, internal workings of other parts of the Windows operating system. These were said to be valuable trade secrets.

This was a ‘front-line’ issue, fundamental to the case, which was argued strongly by both sides. This will be elaborated in the following chapters.

Technical Points and Competition

The competition choice involved balancing Microsoft’s interest in exploiting its server technology for itself and the interest of the wider community in being able to create new server products by being able to interoperate adequately with Microsoft’s server technology.
The function of this chapter has not been to reach any conclusion about the application of the competition rules, and none are expressed. However, three technical comments may help readers to reflect on this.

First, the technology underlying Active Directory was complex and its development had almost certainly required considerable effort. The product was successfully implemented in many organizations and could scale to accommodate large numbers of users, employing a multi-master replication technique to achieve a distributed service.

Secondly, in an existing Windows environment, setting up a server running a Windows operating system was a relatively simple process compared to setting up a server running a non-Microsoft operating system in the Windows environment and that was subject to the limitations described.

Thirdly, while there was a great debate about interoperability, it was not true that Microsoft had to tolerate any copying of its source code. The remedy was intended to require Microsoft to provide sufficient protocol documentation to enable a potential competitor to make a server that could be inserted into a network of Windows work group servers and perform on an equal footing with them.

3. TYING: WINDOWS MEDIA PLAYER

Necessary Background

The reader will be relieved to know that the technological aspects of this second aspect of the case, tying, are very much easier to grasp than interoperability. Furthermore, much (but not all) of the basic, technical 'vocabulary' has already been covered. Before tackling the media player, it is therefore necessary to provide a little background knowledge in the form of a simplified description of the basic media technologies involved. This needs to be supplemented with a brief explanation of the remaining concepts, such as codecs, container files, streaming protocols and digital rights management.

What are Digital Media?

The process of recording images and sound in a digital form involves sampling. When sound waves impinge upon a traditional microphone, they create an electrical signal that changes over time as a function of the minute air-pressure variations involved. This signal is an analogue form of the sound experienced. The reverse process can be readily imagined.

In this, a loud-speaker or iPod earpiece, for example, will take such a signal and vibrate the air as a function of its time variation, thus recreating something close to the original sound. In simple terms, devices like old-fashioned tape-recorders effectively recorded the signal from a microphone. On playback, the tape-recorder's loud-speakers converted the recorded signal into sound.

In the digital world, a different approach is taken to capturing a microphone's output. Here, the process involves sampling. At regular but extremely short intervals, fractions of a second, the analogue signal from the microphone is measured. These measurements are then recorded and, self-evidently, form a sequence of numbers. For this technique to work in practice, it is important that the measurements are made sufficiently accurately and quickly. This entire process is called, unsurprisingly analogue-to-digital (AD) conversion. The opposite process, digital-to-analogue (DA) turns the numbers back into a time-varying signal again. For this to work perfectly, the measurements have to be delivered to the DA converter at exactly the same rate as they were collected.

It turns out that, by sampling at a rate that is greater than twice the highest frequency of the analogue signal, no information is lost. The basic, standard sampling rate for an audio signal is 44.1 kHz. If each sample takes up 16 bits (0s and 1s that can represent 65,536 different signal levels), a quick calculation shows that an hour of stereo music for two independent left and right signals will generate 635 Mbytes of data.

The same line of argument works for video, except that the numbers are much larger. To understand this conceptually, think of a video-camera's focal plane as being three rectangular arrays of light sensors instead of microphones, the array being 1,920 wide by 1,080 high. There are three arrays because there is one for each colour: red, green and blue. Now imagine each sensor collecting 10-bit samples (representing 1,024 different levels) 60 times per second for each of the colours. Another quick calculation shows that this will generate 1.7 terabytes of data for an hour of video. If this were transmitted over a 1 Mbit/sec Internet connection, it would take more than 11 weeks to download a single half-hour television show.

What is a Codec?

Clearly all of the above volumes of sampling measurements are neither an economical nor a practical way of distributing media. To overcome this problem, a large number of data-compression schemes have been invented. These reduce the large volumes of data to a manageable size and yet lose almost none of the original information. They compress the
data using specially-designed algorithms and then store it. Some have been agreed by standards bodies, others are proprietary schemes. Each compression technique also pairs with the reverse process, decompression. These schemes have been given the generic name of 'codecs'. This stands for coder/decoder or compressor/decompressor. Some codecs are able to recreate exactly the original list of signal measurements. These are known as 'lossless codecs'. Others perform less well, but recreate a set of samples such that, following conversion into audio or video, the imperfections generated are barely perceptible to the human ear or eye. These are known as 'lossy codecs'. The latter achieve significant compression rates. Audio and video require separate codecs for greatest efficiency because of their very different spatial and temporal compression requirements.

Files, Formats and Media File Standards

Before proceeding further, it is necessary to understand a little bit about one of computing's basic concepts: files. In simple terms, a file is simply a collection of data that is given a name. It can consist of a collection of records, each of which conforms to a predetermined, common format. It can also contain an arbitrarily structured set of 0 and 1 bits corresponding to an image, for example. The file format can be specified in an arbitrary manner by a programmer for the convenience of a particular program. However, the format can also have been predetermined by an industry body so that files can be freely exchanged between parties and so that specific, pre-agreed data representing particular functionality can be carried. A file is the computer equivalent of the familiar, paper file used in offices. A file can also hold other files. Operating systems have to deal with a large number of files coming from various sources and, of course, files are listed in the directories described earlier.

Some file formats have been designed for a particular type of data such as a photograph or image. The well-known JPEG is an example of this. Others have been designed for different purposes, such as text files for word-processing. In order that a computer can deal with a file in an appropriate manner, there has to be a way of indicating in advance what the file format is. A popular way of doing this is to append the file type as a so-called 'file extension' after the file name and to place a full stop before the extension. Thus a photograph file might be called "Washington.JPEG". Some file formats have also been designed that can be used to hold several types of compressed data produced by codecs. This is called a 'container' file. These containers can each contain several different files within them. Some container file formats are confined to audio, some to single images, whereas others are specifically for multi-media content. Thus, a container holding a movie might contain several separate files intended to be played together such as video, audio and titles. Examples of container file formats are WAV for audio only, TIFF for still images only and AVI for multimedia. Container files are a frequent source of confusion. For example, an AVI container could hold several files to deliver a simple media experience. However, one particular AVI container could well hold files compressed with quite different video and audio codecs to those contained in another. This is because the AVI container has been designed to be able to hold, as part of its constituents, different types of compressed files. By contrast, a non-container format such as a WMA file only holds data compressed by a WMA codec.

In summary, standard formats come from two main sources. Some, such as MPEG, were developed by industry bodies (MPEG stands for Moving Picture Experts Group). Others, such as AVI, were created by private companies, in this particular case Microsoft. There are many different formats and all have various advantages and disadvantages. Different file formats each need their own, specific codec.

This conveniently brings us to the media player.

What is a Media Player?

A media player is an application program. Its basic purpose is to read files of audio and video information, to process these and to output instructions via the operating system that will ultimately be understood by a computer's loud-speakers and displays. A media player runs on a PC client. A key function is to take compressed files and to decompress them in order to play the content. A fully-featured media player must be able to handle different container and codecs in order to be able to do this. Without having the necessary codecs installed, a media player will be unable to play files recorded in a particular format.

A media player, like any application program, makes use of the 'infrastructure' provided by the operating system. It expects that the media containers supported by the operating system will have run-time library support. It expects that the most common codecs will be supported, either being provided with the operating system or being installed later, probably through downloading. The codecs and container libraries (software modules designed to deal with container files) will have had to be written to use general graphics and audio software components of the operating
system that ultimately use the device drivers that operate the specific display and audio hardware.

**Streaming and Downloading**

Basically, audio-visual files can be played in two ways: from a previously downloaded file or from a continuous stream of file data.

With the first possibility, a file is transferred over a network onto a user's PC. This is called a download. Once the download has been completed, a media player then takes the file and plays the content to the user. Since only the transfer of a file from one computer to another takes place before the media player begins its work, no special audio-visual protocols are involved with the transfer.

With the second possibility, the media player takes the data as it is transmitted and plays it back 'on the fly'. Effectively, the file is sent to the media player in a nearly constant stream. This usually involves a specialized streaming-server sending the data in the first place. Streamed data may be sent over a network using special streaming protocols, where the Internet is involved. A streaming protocol trades timeliness for accuracy. The explanation for this is the following. Ordinary file copying is handled by protocols that involve copious acknowledgement of receipt of the data that is transferred, coupled with the retransmission of corrupted blocks of data. However, if a user were viewing a live event, the delays and corresponding 'jerks' provoked by retransmission of any error blocks could easily become unacceptable. Streaming protocols fix such problems. They can, for example, contain so-called 'forward error correction data', which can be used to correct some errors without any retransmission, and for which the sender will not be expecting to receive any acknowledgement. A sender might also receive back and be expected to comply with requests to change the streaming-rate when a particular network connection is unable to keep up. As a result, a media player needs to have streaming capability built in; otherwise, it will be unable to perform the streaming form of playback that uses special protocols. The live streaming of such content (which may be prerecorded) over the Internet is known as webcasting.

**APIs Again**

Software developers often write programs that request an installed media player to provide a particular service. An example of this would be calling a media player API in order to play streamed audio. By doing this, developers avoid having to write new software to create the specific functionality required.

This means that an installed media player has to make APIs available for the programmers and they, in turn, need to have an accurate description of how to use them. To make this easier, suppliers often provide so-called software development kits (SDKs). These are a form of programming tool-box that is used to create software that will automatically use the correct APIs in the appropriate way.

**What is DRM?**

Digitally stored media can be easily copied or transferred from one computer to another. The copying process is accurate and audio-visual content is thus not degraded. This means that such media could be unlawfully copied and widely distributed without any loss of quality occurring. To prevent this, a technological development called digital rights management (DRM) came about. The idea in principle is that special software converts audio-visual content into a protected form before it is made available for distribution. Only media players that support a particular DRM technology are able to convert such protected content into a readable form. Furthermore, media players can only read media-file content if users provide the correct DRM permissions.

Content providers can thus package their material in a DRM-protected form. To this end, suppliers' DRM capability is added to their SDKs. For example, where Microsoft media formats were involved at the time, content providers could use Windows Media Rights Manager for this purpose. The latter was one of the components of the Windows Media SDK.

**Media Player Examples at the Time of the Case**

Media players were produced by several suppliers. They included RealOne Player by RealNetworks, Winamp by Nullsoft, Apple's QuickTime player and Microsoft's Windows Media Player. There were several others. Many depended on codecs supplied by third parties to enable them to process particular file formats. Not all media players supported all formats.

At the time of the case, Microsoft's latest player was Windows Media Player 9. This, for example, did not support Apple's or RealNetworks' formats. Similarly, it did not support the MPEG-4 file format for video content. It did have Microsoft's DRM technology built in as part of a service called Windows Media Rights Manager, mentioned above.

At the server end, streaming content was delivered in the case of Microsoft by software called Windows Media Services 9 Series. This would only run on servers running Windows operating systems.
**Incorporation of Windows Media Player in the Operating System**

At the time of the complaint about conduct in 1999, with one exception, users' media players had been installed on their PC clients in either or both of two main ways: non-Microsoft players had already been pre-installed on their computers by the time of purchase or users subsequently downloaded players over the Internet, paying the appropriate fee where necessary.

The exception was Windows Media Player where a PC ran a Windows operating system. Beginning with Windows 98, Microsoft provided its media player along with the operating system as part of a 'bundle' (bundle is the competition term for tying together separate items). Thereafter, Windows Media Player was always provided with the operating system. As time went by, parts of the Windows Media Player software were effectively removed from the application and became internal components of the Windows operating system. Thus, instead of the operating system having to request the media player to perform some services, the software components for this already resided there. In this way, when users looked up files containing photographs, for example, and requested a so-called 'thumbnails' view that displayed tiny images of the visual content stored within them, these absorbed media player components were used by the operating system to generate the images.

This meant that the remaining Windows Media Player application could not run without an underlying Windows operating system.

This characteristic of the close relationship between the Windows Media Player and the Windows operating system is important to the understanding of the Microsoft case.

By contrast, at the time, Microsoft had released a self-contained version of Windows Media Player 9 that would run on Mac OS, Apple's operating system, which was, in turn, based on UNIX. Self-evidently, the underlying Apple operating system did not already incorporate parts of the Windows Media Player code. This Apple version of the latter effectively reverted it to a standalone application.

**Windows Media Player Redistributable Code**

At the time of the case, a downloadable form of Windows Media Player 9 was made available by Microsoft. This consisted of a media player in modular form and was called Windows Media Player Redistributable.

The idea for having this was as follows. For some specialized purposes, Microsoft would supply an operating system that was stripped down to its bare essentials. This could involve the Windows Media Player not being present. In such circumstances, should a software developer need to introduce parts of or even the entire media player functionality, he or she could download and then incorporate the necessary components available from Windows Media Player Redistributable. This also made it possible for content providers to include Windows Media Player 9 in their webpages or for software developers to integrate it into their applications programs. Clearly, to exploit the latter, developers would make use of the media player APIs that the various components exposed.

All this meant that if a version of the Windows operating system had to be produced without the Windows Media Player, the latter could quickly be reintroduced, even by applications programs or webpages.

In the Microsoft case, arguments relating to the removal or not of Windows Media Player 9 were also considered to cover implicitly the redistributable code version. The role of the redistributable code is, therefore, important to the understanding of the Microsoft case.

**Technical Point Summary: Relevance to Tying**

To bring this section to a close, it is worthwhile summarizing the preceding explanations.

First, it was possible to create a standalone version of Windows Media Player 9 that was independent of the Windows operating system.

Secondly, it was also possible to have a version of the Windows operating system without Windows Media Player 9.

Thirdly, in such a case, it was even possible to reinstall modules of Windows Media Player 9 from Windows Media Player Redistributable.

4. **BRIEF COMPARISON WITH THE TECHNOLOGY ISSUES INVOLVED IN THE US MICROSOFT CASE**

Having understood the basic technology issues involved in the EC case, this is now a convenient point at which to compare briefly what was involved in the US proceedings.

**Use of the Term ‘Middleware’**

The US case concerns a category of software to which the label ‘middleware’ was attributed. This was an unusual use of this term. In simple terms, middleware is normally used in the information technology world to describe the unseen software ‘glue’ that enables distributed applications to work. Parts of these may be running on one or more computers connected
via a network. Middleware can be thought of as a set of services which are accessed by programmers through programming interfaces. Middleware services enable the information processing functions of an application to be transparently distributed across a network. Programmers can access such services by using their appropriate APIs without needing to know where the services will be performed. Thus COM/DCOM mentioned earlier is an example of conventional middleware.

The US case concentrates on browsers (specifically Internet Explorer and Netscape Navigator) and on a machine-independent language, Java. Both the Internet Explorer application and the Java language can be used as platforms by other applications to provide their desired results. As such, in the US case, the term 'middleware' has been somewhat unconventionally used to describe both collectively. In fact, in the Final Judgment, the term is used to cover specifically the functionality provided by Internet Explorer, Microsoft's Java Virtual Machine, Windows Media Player, Windows Messenger, Outlook Express and their successors or software having similar functionality.

Navigator and Java Provided Potential, Operating-System-Independent Platforms

In simple terms, a browser is a program that is used to locate and display webpages. The primary use of a browser is to perform this function for material available on the World Wide Web. It is also used to access information on web servers connected to private networks, as well as files stored in an appropriate manner within file systems.

As part of its functioning, a browser takes HTML (HyperText Markup Language), and renders it into the corresponding characters and format within the browser window. Browser functionality is also frequently enhanced by so-called 'plug-ins'. These are software components that usually do not work in a standalone fashion but rely on the services provided by the application to which they 'plug in'. These allow an application to display in a browser window content that is much richer than could be obtained from the browser alone. Flash is an example of a plug-in.

Netscape was originally developing versions of its Navigator browser to run on different operating systems. If a software developer were to create an application that entirely relied on the APIs provided by Navigator, then in a networked environment, that application would be able to provide its results, through Navigator, on many different operating systems.

Java consists of the following and was originally developed by Sun:

- programs are written in the Java language;
- the Java instructions are translated by a program called a compiler into an intermediate language called Java bytecode;
- another program called a Java Virtual Machine (JVM) converts the bytecode into machine code, which is then executed.

A JVM is created that is specific to an operating system and the processor on which it is running. There are thus several JVMs, each one corresponding to a different operating system/processor. Part of this Java environment includes a set of shared libraries of commonly used routines accessed via APIs. It can thus be seen that if a Java program could be entirely supported by a Java library, it would be able to run on any machine and also be independent of the underlying operating system.

In 1995, Netscape agreed to provide a copy of the Java environment with each copy of its Navigator browser.

Integration of Internet Explorer into the Windows Operating System

Originally, Internet Explorer was provided in 1995 as a standalone application for Windows 95. Standalone versions were also created for other operating systems, including Mac OS.

Subsequently, Microsoft placed part of Internet Explorer's specific code into the operating system, in particular into the libraries providing operating system functions. In this way, Internet Explorer would not function without an underlying Windows operating system. This is an analogous process to that described in detail previously concerning Windows Media Player. It also worked the other way around. For example, if a software developer decided to use Microsoft's HTMLHELP function to implement an application's help function, he or she would invoke the requisite API. However, this was only available through Internet Explorer.

Java Developers Obliged to Depend on the Windows Environment

Sun's original intention was that software developers would be able to create programs that would run on any computer and irrespective of the operating system. There was, however, a problem. The Java libraries were somewhat limited and did not expose sufficient APIs to enable a normally featured program to work. As a result, programmers had to make up this deficiency by relying on APIs exposed by the specific operating system on top of which the Java environment was functioning. This meant that the original purpose of Java was not being achieved.
To overcome this, Sun sponsored the development of a mechanism that would make it possible to use these operating-system-specific APIs but which hid this from a Java programmer. This would then make Java programs truly portable between different operating systems. The mechanism was called Java Native Interface (JNI).

The Java environment supplied by Netscape along with Navigator complied with Sun’s mechanism.

Microsoft developed its own version of Sun’s mechanism and which executed more rapidly. This was called Raw Native Interface (RNI). However, there was an incompatibility. A developer who produced software using Microsoft’s mechanism rather than Sun’s would find that the software would not run on Sun’s Windows version of the JVM. This incompatibility was removed in 1998, when a court order required Microsoft to implement Sun’s process.

Subsequent to this, Sun developed a technique called Java Remote Method Invocation (RMI). However, Java RMI was not provided with Microsoft’s Java environment. This created a problem for programmers who would normally call communications interfaces that certainly had to be available on end-users’ Windows client PCs. To get around this, they were obliged to rely on Microsoft-specific interfaces.

In 1998, Netscape ceased to distribute Java with its latest version 5 of Navigator. By contrast, a standalone version of Microsoft’s Java environment was made available to independent software developers. It was provided independently of Internet Explorer.

Eventually, the only effective way of being absolutely sure that a developed application would be compatible with Microsoft’s Java environment was to use Microsoft’s development tools.

The US Remedy

The Final Judgment in the United States required that Microsoft document the Windows APIs and communications protocols that were used between client and server operating systems. The reason for this is obvious from the above brief description of the technical aspects of the US case. The browsers and the Java environment – the middleware – were running on client PCs. The applications serving them were running on servers. It contrasts with the EC case requiring that server-to-server protocols be documented also.

As opposed to the Commission’s Decision concerning the Windows Media Player, it only required the means of access, in whatever form, to the offending ‘middleware’ product to be removed. It did not require removal of the code from the Windows operating system.

5. CONCLUSIONS

We, therefore, arrive at a state of knowledge that enables this chapter to draw to a close.

At this point, the reader is well-equipped to follow the numerous arguments made by the parties involved in this case. Many of the technical points explained by the various participants during its course went considerably beyond what has been provided here. However, with the foundations provided by this chapter, the essentials of what is at stake should be comprehensible.

Sun’s original request for ‘the complete information required to allow Sun... to provide native support for the complete set of Active Directory technologies on Solaris’ and information ‘necessary to provide native support for COM objects on Solaris’ should now be understandable as should the limitations of the various solutions that were proposed. The manner in which Windows Media Player had become linked to the Windows operating system should also be clear.

Furthermore, the terminology of Remedies laid out in the Decision should now be understandable. For example, in the case of interoperability, ‘Microsoft should be ordered to disclose complete and accurate specifications for the protocols used by Windows work group servers in order to provide file, print and group and user administration services to Windows work group networks... Microsoft should not be required to disclose its own implementation of these specifications, that is to say, its own source code.’ In the case of tying, ‘Microsoft will have to offer a version of Windows for client PCs which does not include Windows Media Player... Microsoft must refrain from using any technological... means which would have the equivalent effect of tying WMP to Windows. The unbundled version of Windows must in particular not be less performing than the version of Windows which comes bundled with WMP, regard being had to WMP’s functionality which, by definition, will not be part of the unbundled version of Windows.’

In summary, the technical background has been laid down for the chapters that follow.

As a final comment, it should now be clear why the technical complexity inherent to this dispute strained the abilities of judges, advocates and experts to unprecedented limits as they struggled to understand, explain and articulate the issues in what is, perhaps, one of the most unusual competition cases ever.