EXHIBIT B

UNITED STATES DISTRICT COURT
NORTHERN DISTRICT OF ILLINOIS
EASTERN DIVISION

EOLAS TECHNOLOGIES, INC.,

Plaintiff,

v.

MICROSOFT CORPORATION,

Defendant.

No. 99 C 0626
Judge James B. Zagel

MEMORANDUM OPINION AND ORDER

Ease of interactivity is a key ingredient to the popular success of the Internet and World Wide Web. From an end-user perspective, manipulating data in a seamless web page, harnessing the computing power of a network of machines, and enjoying the luxury of an easy-to-use browsing experience are all benefits of the current state of today's computer technology. For many end-users, this interactivity comes shrouded in mystery; the underlying technology, computer codes, and indeed, the history of computer networks become relevant only on a need-to-know basis. In this sense, computers differ little from automobiles and microwave ovens with frozen entree sensors. This case unveils a small portion of Internet technology.

Michael Doyle, David Martin and Cheong Ang invented a method for building a web browser that can display interactive objects embedded in a single web page and that uses another application to enable the interactivity.[1] In this case, plaintiff Eolas Technologies is the exclusive licensee of the Doyle patent, U.S. Patent No. 5,838,906 (issued November 17, 1998) (the '906

[1] I describe the invention in general terms for now.

Patent). Eolas alleges that Microsoft Corporation infringes that patent. However, infringement and the particular accused products are not at issue yet. At this time, the only question presented is the legal question of claim construction.

A patent claim is that portion of the patent document that defines the invention, and must "particularly poin[t] out and distinctly clai[m] the subject matter which the applicant regards as his invention." 35 U.S.C. § 112. The words used in describing the invention are often subject to interpretation, and in this case the parties dispute the meaning of some terms used in the '906 Patent. The meaning of the words, the construction of the claims, is for me to decide and it must be decided before the issue of infringement. *Markman v. Westview Instruments*, 517 U.S. 370, 391, 116 S.Ct. 1384, 1396 (1996).

I. Background

A. The Internet and World Wide Web

Although issued in 1998, the '906 Patent takes us back to 1993 and the early days of the World Wide Web. Then, as now, the Internet referred to a vast network of networks. A typical network consists of client computers (*e.g.*, individual desktop computers) that request information from server computers that, in turn, provide information back to the client. With the use of a uniform standard, the Transfer Control Protocol/Internet Protocol (TCP/IP), localized networks are able to communicate with other local area networks, creating a geographically diverse network – the Internet.

Other standards, HyperText Transmission Protocol (HTTP) and HyperText Markup Language (HTML), allow for the transmission and exchange of data among computers in a specific format, called hypertext (or when images, video or sound are involved, hypermedia).

Hypertext documents are located throughout the Internet and are identified by reference to their addresses, known as URLs (Uniform Resource Locators). Hypertext documents usually are displayed on a user's screen by way of a software program called a browser; the browser reads (parses) the hypermedia document (written in HTML) and displays (renders) it accordingly.[2] The HTML code may identify links to other hypertext documents and the browser displays those links. The user can click on the link, and, using HTTP, the browser retrieves that next hypertext document to render. The array of HTML-based hypermedia documents (web pages), linked together and navigable across geographically diverse networks, is called the World Wide Web.

Notwithstanding the "digital divide" that separates many people in this country (and the world), I assume familiarity with these terms and with the basics of computing in the Internet environment.

B. The Patented Invention

The '906 Patent is entitled "Distributed Hypermedia Method for Automatically Invoking External Application Providing Interaction and Display of Embedded Objects Within a Hypermedia Document." The inventors identified two problems that they sought to solve. First, from the individual end-user perspective, hypermedia documents were limited in their ability to deal with large data objects (*e.g.*, particularly large video files, pseudo-3-D animated sequences and the like) because of bandwidth constraints and the limited processing power of one person's computer. For example, an animated sequence often entails receiving data at a rate

[2] In the early 1990s, Mosaic was the state-of-the-art browser; today, Netscape's Navigator and Microsoft's Internet Explorer are the browsers most familiar to consumers.

fast enough to display at 30 frames per second and requires a computer with a sophisticated processor able to perform the calculations necessary to animate images. As the inventors stated in their patent, "Today's browsers and viewers are not capable of performing the computation necessary to generate and render new views of these large data objects in real time." '906 Patent, col. 5, ll. 53-55.

Second, the inventors noted the limited capability of the state-of-the-art browser (Mosaic) to provide interaction with data objects. Generally, the user needed to go outside the browser to interact with the data. "Users are limited to traditional hypertext and hypermedia forms of selecting linked data objects for retrieval and launching viewers or other forms of external software to have the data objects presented in a comprehensible way." '906 Patent col. 6, ll. 35-39. At the time, these external software programs were commonly called helper applications. When Mosaic encountered web pages that required helper applications to deal with objects, it forced the user to interact with a separate display area (a pop-up window). There was no communication between helper and the browser, *i.e.*, the browser was inactive while the helper was active. '906 File History, Paper # 19, pp. 7-8.

The inventors envisioned "a system that allows a user at a small client computer connected to the Internet to locate, retrieve and manipulate data objects when the data objects are bandwidth-intensive and compute-intensive;" and that allows "a user to manipulate data objects in an interactive way to provide the user with a better understanding of information presented." '906 Patent, col. 6, ll. 40-47. An example of this idea is a browser that is capable of displaying a web page that retrieves complex 3D medical images (*e.g.*, an image of an embryo). The computational processing of the image is done by more powerful computers located

remotely from the user's, meanwhile the browser allows the user to manipulate the image (rotate, scale, reposition the viewpoint) without exiting the browser display area. '906 Patent, col. 7, ll. 7-28.

The Patent describes a preferred embodiment of the invention (a modified Mosaic browser), and outlines in some detail a sequence of events that exemplifies the method. In the example, an end-user has a client workstation. A browser and at least one other application reside on the client's hard drive. The browser loads a hypermedia document and begins to parse its HTML code. The HTML code indicates that an external application is required to process some embedded (*i.e.*, in the page) object.[3] The browser calls on that application, invokes it (without further input from the user), and the application processes the request based on information it receives from the browser, the user, and perhaps other resources on the Internet. In the example, the application is an image viewer that processes and displays 3D images (the data object) and allows the user to manipulate the images. The application's displays are integral to the browser's -- a 3D image of an embryo appears in the browser display area and the user is able to manipulate the image immediately.

I refer to this example merely to illustrate the inventors' idea, not to define with precision the scope of the invention. For that, the devil is in the details and I must first consider the words

---

[3] HTML code is text-based. It uses words to tell the browser how a web page is to be displayed, describing colors, fonts, placement of text, etc. The format of an HTML file uses structures called tags, which are descriptive words placed in angle brackets that convey instructions to the browser, *e.g.*:

```
<EMBED
        TYPE = "type"
        HREF = "href"
        WIDTH = "width"
        HEIGHT = "height"
>
```

'906 Patent col. 12, ll. 54-65.

used to define this invention -- the words in the claims, not the descriptive background exposition of the Patent. The Patent includes two independent claims, Claim 1 and Claim 6. Both claims use the same terms and I follow the parties' convention by focusing on Claim 6.

C. Claim 6

Claim 6 provides, in its entirety:

> What is claimed is: [...]
> 6. A computer program product for use in a system having at least one client workstation and one network server coupled to said network environment, wherein said network environment is a distributed hypermedia environment, the computer program product comprising:
> a computer usable medium having computer readable program code physically embodied therein, said computer program product further comprising:
> computer readable program code for causing said client workstation to execute a browser application to parse a first distributed hypermedia document to identify text formats included in said distributed hypermedia document and to respond to predetermined text formats to initiate processes specified by said text formats;
> computer readable program code for causing said client workstation to utilize said browser to display, on said client workstation, at least a portion of a first hypermedia document received over said network from said server, wherein the portion of said first hypermedia document is displayed within a first browser-controlled window on said client workstation, wherein said first distributed hypermedia document includes an embed text format, located at a first location in said first distributed hypermedia document, that specifies the location of at least a portion of an object external to the first distributed hypermedia document, wherein said object has type information associated with it utilized by said browser to identify and locate an executable application external to the first distributed hypermedia document, and wherein said embed text

> format is parsed by said browser to automatically invoke said executable application to execute on said client workstation in order to display said object and enable interactive processing of said object within a display area created at said first location within the portion of said first distributed hypermedia document being displayed in said first browser-controlled window.

D. The Disputed Terms

The parties dispute the meaning of the following key phrase: "**wherein said object has type information associated with it utilized by said browser to identify and locate an executable application.**" What is an executable application? What is the type information that must be associated with the object? What does it mean for the type information to be utilized by said browser to identify and locate the executable application?[4]

E. The *Markman* Hearing

In order to resolve this dispute over claim construction, I held a hearing pursuant to *Markman v. Westview Instruments*, 517 U.S. 370, 116 S.Ct. 1384 (1996). The parties submitted briefs, the Patent itself, and the entire File History (the official record of proceedings in the Patent Office leading up to the issuance of the Patent). In addition, both parties offered expert testimony. The plaintiff offered the testimony of Edward Felten, Associate Professor of Computer Science at Princeton University. The defendant offered the testimony of H.E.

[4] In addition, there are additional claim terms and phrases subject to possible dispute: "browser application;" "distributed hypermedia environment;" "distributed hypermedia document;" "to enable interactive processing;" and "automatically invoke." To a lesser degree there may be a dispute as to the meaning of "program code for causing" and "text format." However, the scope and relevance of the dispute on all these terms is unclear. Microsoft suggests I defer construction of these terms and I agree. The parties are free to brief and argue in detail the construction of any other claim terms in light of this opinion.

Dunsmore, Associate Professor of Computer Science at Purdue University, and Michael Wallent, Product Unit Manager for Internet Explorer.

Unless specifically referenced in this opinion, I considered only the live testimony adduced at the hearing, the patent itself, and the file history. No other evidence is admitted.

II. Discussion

A. The Standards for Claim Construction

The basics of claim construction are well-settled. This is an issue of law for the court to decide. *Markman*, 517 U.S. 370. In interpreting the meaning of a claim, the focus is on intrinsic evidence – the claims, the specification and the prosecution history. *Vitronics Corp. v. Conceptronic, Inc.*, 90 F.3d 1576, 1582 (Fed.Cir. 1996). This intrinsic evidence constitutes the public record of the inventors' claims; it puts the public on notice. A competitor should be entitled to examine this public record and understand the scope of the claimed invention. *Id.* at 1583.

However, I have received some extrinsic evidence in the form of opinion testimony from experts skilled in the art of computer science. Given the nature of the invention, this testimony is necessary so that I may understand the technology and construe the claims according to their ordinary and plain meaning, as understood by one skilled in the art. *Pitney Bowes, Inc. v. Hewlett-Packard Company*, 182 F.3d 1298, 1309 (Fed.Cir. 1999); *Interactive Gift Express, Inc. v. Compuserve Inc.*, 231 F.3d 859, 2000 WL 1644598 *1, * 6 (Fed. Cir. 2000) (the viewing glass through which claims are construed is that of a person skilled in the art).

B. "Executable Application"

The parties agree that a key part of the invention is the ability of the browser to automatically invoke some external program (vis-à-vis the hypermedia document) to process the data object. Generic examples are image viewers, word processors, and spreadsheets – programs that display and allow the user to interact with data. The claim refers to this program as some "executable application."

The dispute is over the scope of "executable application." Eolas defines this term as "program code for causing the display of the object and enabling interactive processing of that object." Plaintiff's Memorandum In Support of Claim Construction, p. 16. Microsoft proposes that executable application refers to "standalone" programs. The limitation of standalone, as used by Microsoft, means that the program can be executed (launched, run or started) irrespective of whether any other programs have been launched or are running. Defendant's Initial Brief on Claim Construction Issues, p. 8.

Computer code is often bundled into discrete components. These components can perform specific functions and be used as building blocks for larger programs. Some components exist separately from larger applications, and are summoned to assist a larger program (sometimes called a host program) when needed. Dynamically linked libraries (DLLs) are types of components that can be shared by different applications to perform common functions. The example used by the parties is a spell checker. Both word processors and spreadsheet programs offer spell checking capability and can share one spell checking DLL that exists as a separate block of code from the larger programs. Components like DLLs must be invoked by some other application, they cannot be executed without some host. Thus, in the

spell checker example, one cannot run a spell checker unless another application, the word processor or spreadsheet, is also running.

Microsoft's proposed construction of executable application excludes components, such as DLLs, from the scope of the term. To be standalone, an application cannot be dependent on another application. Microsoft does not believe a component (a routine, a library, or a module), hosted by the browser to perform some function, could be the executable application referenced in the invention. The browser and the executable program must be independent of each other, and function as peers. Eolas says the executable application could be a DLL or some component, as long as it is code that can be launched and enable interactive processing (*i.e.*, allows the user to do something to or with the data, which is the point of the invention).[5]

Does "executable application" have a plain and ordinary meaning to someone skilled in the art of computer science? Apparently not. Professor Felten defined it as "a sequence of computer instructions in a format that is capable of being executed." Felten Report ¶ 58.[6] Professor Dunsmore defined it as "a standalone program that can be run without needing to be included in some other program." Dunsmore Direct Examination, October 26, 2000, Tr. at 201.[7] This difference in opinion is resolved by examining which definition best captures the inventors'

---

[5] I note that there may be a dispute as to the scope of "interactive processing," *see* note 4, *supra*., but I understand Eolas's position to be that the executable application must be able to manipulate the data object (edit numbers, rotate images, etc.).

[6] During his live testimony, Prof. Felten did not provide a definition of executable application. Instead, he opined in a defensive posture, saying that an executable application did not have to be standalone. I admit ¶ 58 of the expert report to provide Felten's baseline, affirmative definition.

[7] At this writing, the transcript of proceedings has not been officially certified. Citations are with reference to an unofficial transcript provided to the Court.

use of the term (and related concepts) in the patent itself and the file history.[8] It is the intrinsic evidence, after all, that is generally dispositive of claim construction issues, and must put the public on notice as to how the inventors are using the terms. *See Vitronics*, 90 F.3d at 1582 (patentee may choose to be his own lexicographer); *see also Pitney Bowes*, 182 F.3d at 1311 (term must be read to correspond to meaning in context); *see also Toro Co. v. White Consol. Indus., Inc.*, 199 F.3d 1295, (Fed. Cir. 1999) (patent documents establish usage of words in connection with claimed subject matter).

The term "application" does appear to have an ordinary meaning in the art of computer science. In 1994, the Microsoft Press Computer Dictionary defined application as "a computer program designed to help people perform a certain type of work. An application thus differs from an operating system (which runs a computer), a utility (which performs maintenance or general-purpose chores), and a language (with which computer programs are created)...." Microsoft Press Computer Dictionary 23-24 (2nd ed. 1994). A few years later, the same dictionary defined application as "a program designed to assist in the performance of a specific task, such as word processing, accounting, or inventory management. *Compare* utility." Microsoft Press Computer Dictionary 27 (3rd ed. 1997). Finally, a third (non-partisan) dictionary offers the following definition: "A program or group of programs designed for end users." ZD Webopædia, at http://www.zdwebopedia.com/TERM/a/application.html. I read these three definitions to mean that an application is a computer program, that is not the

[8] I am not surprised the two experts disagreed on the general meaning of executable application. Computer science does not yet seem to enjoy intra-discipline agreement. For example, in 1990 (ancient history from the perspective of this case) one writer noted that "stand-alone code is program code which *does not* enjoy the full status of an application." Craig Prouse, "Technote PT 35: Stand-Alone Code, ad nauseam," (August 1990) (emphasis added) (with reference to Apple Macintosh), at http://devworld.apple.com/technotes/pt/pt_35.html. I note this solely to highlight the context-dependent nature of computer science terminology.

operating system (OS) or a utility, that is designed to allow an end-user to perform some specific task.[9] Dictionaries are extrinsic evidence, but may be considered alongside intrinsic evidence. *Interactive Gift Express*, 231 F.3d 859, 2000 WL 1644598 at * 6 n.1.

1. The Claim

It is clear from the claim language that whatever an executable application is, it must have certain features. It must be external to the hypermedia document, it must be located on the client workstation, and it must allow the user to interact with data. A component could have these features. Thus, the functions enumerated by the claims do not necessarily imply an exclusion of components. The claim language provides no other guidance.

2. The Specification

The claim mentions two applications – the browser and the executable. In the specification, the inventors stated that the browser's functionality (the invention) could be implemented using "routines, processes, subroutines, modules, etc." '906 Patent, col. 13, ll. 60-62. I agree with Microsoft that, at a minimum, this language stands for the unremarkable proposition that the browser can be built by putting modules together. At the hearing, both experts agreed that components can be used as building blocks for larger programs. The reference to modules (of which DLLs are a type) is not a reference to a module as an external application. However, as a person skilled in the art of computer science, Professor Felten read this passage to mean that the browser application could itself be a component or module invoked by another application. Felten Direct Examination, October 25, 2000, Tr. at 59-60. In the

[9] According to the ZD Webopædia "execute" is synonymous with "run" and "launch." ZD Webopædia, at http://www.zdwebopedia.com/TERM/e/execute.html.

context of this invention, if the browser could be a component, then it is a logical inference that the executable application (the only other application referenced in the claim) could be a component as well. As Microsoft points out, this is an inference, not an explicit statement in the specification. Nevertheless, I find it is a logical inference. Moreover, the preceding sentence is an acknowledgment of all methods of programming known in the art ("various programming approaches such as procedural, object oriented or artificial intelligence techniques may be employed"). '906 Patent, col. 13, ll. 57-59. The passage indicates that the patentees sought a broad scope to their invention and tried to foresee all possible programming methodologies that were possible.

Elsewhere in the specification, the inventors stated that the executable application could be installed as a terminate and stay resident (TSR) program. '906 Patent col. 9, ll. 11. TSRs are programs that continue to occupy memory space even after they are terminated (no longer being used). In the event the user wants to use the program again, the computer saves time by being able to run the program without loading it into memory again. The experts debate whether TSRs are standalone or not, but this debate is largely irrelevant.[10] I accept Microsoft's proposition that TSRs are not traditionally considered components of other programs. This portion of the specification does not relate to componentization at all and does not provide a definition of executable application. The context of the specification language with regard to

[10] Prof. Felten takes issue with calling a TSR standalone because he does not characterize a TSR as residing in its own memory space. Felten Direct Examination, October 25, 2000, Tr. at 79. However, from Prof. Dunsmore's testimony at the hearing, I gather Microsoft to focus its definition of standalone on the ability of a program to run without some other program running, and not on OS memory allocation. I believe both Professors Felten and Dunsmore would agree that a standalone program is a program that can run regardless of whether another program is running. *See* Felten Cross-Examination, October 26, 2000, Tr. at 121. This is the definition I use, based on the agreement by two experts skilled in the art.

TSRs is not to limit executable applications to any one permutation, but to comment on the possible use of memory-saving techniques. The inventors did not use the language of limitation.

In discussing the executable application in reference to the preferred embodiment (the 3D embryo imaging browser), the patent specification uses the example of "x-vis," a data visualization tool designed to operate on three dimensional image objects. '906 Patent, col. 13, ll. 3-11. The parties agree that x-vis is not a component. The Patent goes on to say, "However, any manner of application program may be specified by the TYPE element so that other types of applications, such as a spreadsheet program, database program, word processor, etc. may be used with the present invention." '906 Patent, col. 13, ll. 11-15. Eolas emphasizes the "any" and the "etc." of this sentence to suggest that the inventors wanted executable application to have the broadest possible meaning. Microsoft emphasizes the examples listed – spreadsheet program, database program, and word processor – as classic examples of standalone applications.

Microsoft argues that since the preferred embodiment disclosed in the specification does not use a component for the executable application, the patent does not cover such use and indeed, does not teach someone how to use a component in that way. The preferred embodiment launches the executable application as a "child process of the current running process (Mosaic)." '906 Patent col. 15, l. 22. Elsewhere, the inventors repeatedly discuss the communication between the browser and the executable application as "interprocess communication." '906 Patent col. 7, ll.1-4; col. 9, ll. 7-10. The inventors discuss a custom Mosaic/External Application Program Interface, MEAPI, that allows the browser to communicate with the executable application. '906 Patent col. 12, ll. 9-11. Microsoft says MEAPI makes no sense if the executable application could be a DLL, since no such interface

would be required. Moreover, the references to child process and interprocess communication must refer to peer applications, not a component and a host, to make sense. Finally, Microsoft says the program code submitted to the Patent Office does not enable Mosaic to launch an interactive DLL. *See* Felten Cross-Examination, October 25, 2000, Tr. at 125.

There is no doubt that the preferred embodiment does not use DLLs or components as the executable application. Nor is there any doubt that the inventors repeatedly said that the preferred embodiment was but one possibility of the invention in practice. In reading the specification, "care must be taken to avoid reading limitations appearing in the specification into the claims." *Interactive Gift Express*, 231 F.3d 859, 2000 WL 1644598 at * 5 (quotation omitted). The specification language cited by Microsoft does not limit the term executable application, it generally does not even use the term. The Federal Circuit "consistently declines to construe claim terms according to the preferred embodiment." *Northern Telecom Limited v. Samsung Electronics Co., Ltd.*, 215 F.3d 1281, 1293 (Fed. Cir. 2000). Even if the entire specification expresses a preference for one method of practicing the invention over another, that is not enough to limit claim terms. *Id.* In the end, I find Microsoft's reading of the specification to be overly strict. The implementation discussed in the specification does not limit the claims; it teaches what the patentees had in mind, the problems they sought to solve, and an example of how to do it. By acknowledging other possibilities, however, and by acknowledging the general usage of object oriented programming, routines, and modules, the inventors expressly avoided limiting the claims by way of the specification. *See* '906 Patent, col. 16, ll. 48-57. It is true that the patentees sometimes referred to "this invention" in the specification and the Federal Circuit has held that "when the preferred embodiment is described in the specification as the invention

itself, the claims are not necessarily entitled to a scope broader than that embodiment." *Modine Manufacturing Co. v. United States International Trade Commission*, 75 F.3d 1545, 1551 (Fed. Cir. 1996)[11]; *see also Wang Laboratories v. America Online, Inc.*, 197 F.3d 1377, 1383 (Fed. Cir. 1999). But as *Wang Laboratories* makes clear, "although precedent offers assorted quotations in support of differing conclusions concerning the scope of the specification, these cases must be viewed in the factual context in which they arose." *Wang*, 197 F.3d at 1383. The specification in this case clearly states that the preferred embodiment is not the only way of utilizing the invention. Therefore, I do not find the specification to limit the claims to a specific definition of executable application.

The patent claims and specification are focused on function, not jargon. While the examples used (spreadsheets, word processors, etc.) may be traditionally thought of as standalone applications, peers of browsers, I do not read the claims and specification to be concerned with that element of programming or memory allocation. Instead, the inventors simply referenced examples of computer code that can take specific types of data and use them to interact with the end-user. If computer code in the form of a DLL (a programming technique well known in the art at the time of patent prosecution) can be launched by the browser and interact with the user, then it is the executable application contemplated by the claims and specification. Indeed, as Eolas points out, if the executable application must be a peer of the browser, then the passage referencing the implementation of the browser through use of modules supports its inference of executable application as a module as well.

---

[11] Abrogated on other grounds by *Festo Corp. v. Shoketsu Kinzoku Kogyo Kabushiki Co., Ltd.*, – F.3d –, 2000 WL 1753646 (Fed. Cir. Nov. 29, 2000).

3. The File History

"[P]referred embodiments, without more, do not limit claim terms," *Northern Telecom*, 215 F.3d at 1293, and Microsoft says it has more. Any ambiguity in the claim and specifications is resolved by the File History. According to Microsoft, it is clear that the inventors disclaimed the use of components, indeed denigrated their usefulness in prosecuting the patent. Thus, by looking at the proceedings before the Patent Office, Microsoft says I may construe the claims to exclude components or DLLs from the definition of executable application. *See Elkay Manufacturing Co. v. Ebco Manufacturing Co.*, 192 F.3d 973, 979 (Fed. Cir. 1999) (claim limited by arguments made during prosecution history and examiner's reasons for allowance).

Doyle, Martin and Ang did not sail through the Patent Office. The patent examiner rejected the patent three times, saying the invention was an obvious combination of prior art (inventions that pre-existed the '906 application). File History, Paper # 4, pp. 4-5; File History, Paper # 12, pp. 2-5; File History, Paper # 15, pp. 2-4. The relevant statute states: "a patent may not be obtained... if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made...." 35 U.S.C. § 103(a).

The prior art included HTTP, HTML, clients, servers and browsing software (*i.e.*, Mosaic); the inventors understood that these basic Web technologies were prior art and disclosed them to the examiner. At first, the examiner thought it would have been obvious to combine this prior art with the teachings of Hansen's "Enhancing documents with embedded programs: How Ness extends insets in the Andrew Toolkit." File History, Paper # 4, p. 5. However, in rejecting the patent, the examiner agreed with the inventors that the disclosed prior art by itself

"does not have embedded controllable application [executable/ interpretable/ 'launchable' program instructions/ codes] in the hypermedia document." *Id.* Early in the history, the executable application is thus defined as any executable, interpretable or launchable program instructions or codes. This is a broad definition that does not exclude components, and confirms Eolas's view of the executable application. Microsoft says I should not read too much in this language; at the time, the claims used the term "controllable application" not "executable application." *See* File History, Paper # 1, p. 29. Moreover, urges Microsoft, later usage narrowed the term. While it is true that the original claim language used "controllable" instead of "executable," it is clear that the examiner thought those terms essentially synonymous and that is relevant to my inquiry here.

I read the file history to begin with a broad definition of executable application, inclusive of componentization. I next review the subsequent history to glean whether the inventors or the examiner narrowed their view of the term.

a. The Khoyi Patent

The examiner rejected the patent a second time because he felt the invention was an obvious combination of Mosaic and another prior art patent, U.S. Patent No. 5,206,951 (issued April 27, 1993) (the Khoyi Patent). Khoyi teaches an operating system in which functions normally thought to be performed by applications are performed by routines bundled as part of the OS. Applications could then use and re-use these functions; Khoyi teaches a kind of componentization. The inventors submitted arguments to the examiner to persuade him that in fact, their invention was not an obvious combination of Mosaic and Khoyi. The examiner

considered the argument and withdrew his Khoyi-based rejection. File History, Paper # 15, p. 2.[12] Microsoft says this means the inventors disavowed componentization.

Microsoft's reading of the File History on this point is on too high a level of abstraction. Khoyi teaches a kind of componentization, and the inventors argued that their invention was different than Khoyi. However, one must examine the very factual nature of the distinctions that the inventors drew. The inventors did not take issue with the fact that Khoyi allowed a kind of shared functionality across applications. They did take issue with the fact that Khoyi was an operating system, not an application. Doyle, Martin and Ang told the examiner that their invention was not dependent on a particular operating system, while Khoyi clearly was; they told the examiner that cross-platform functionality (*e.g.*, as allowed by the Java Virtual Machine) was something new and different from Khoyi. The inventors thus quoted from the Khoyi patent itself and said "functions and operations which would normally be performed by the application programs themselves, are performed by libraries of routines [pack routines]." File History, Paper # 14, p. 16 (quoting Khoyi Patent, col. 11, ll. 57-59). The inventors noted the difficulty in making Mosaic work with Khoyi, saying "Mosaic would have had to be significantly modified in a number of additional complex and nonobvious ways to achieve that combination." File History, Paper # 14, p. 17. This was not a statement that the '906 invention could not work with components or that Mosaic could not work with components. It was made in the context of noting the limitations of Khoyi's operating system model in a cross-platform world. The

[12] The examiner's withdrawal of the Khoyi-based objection was not on the merits, but instead on mootness grounds. The examiner said that the Khoyi issue was moot since another obviousness ground for rejection existed, the Koppolu Patent. *See infra*, at II.B.3.b.

distinction drawn by the inventors was not between "executable applications" and components, but between applications and operating systems.

The inventors made another argument to suggest a substantive difference between Khoyi-componentization and other means of employing components. Doyle told the examiner to look to Microsoft. Doyle said that Mosaic plus Khoyi (developed by Wang Laboratories) was not the same as Microsoft's ActiveX technology, Microsoft said so itself. File History, Paper # 14, Doyle Affidavit pp. 10-11. Doyle said that this shows that his invention is different than some obvious improvement upon Mosaic + Khoyi because ActiveX uses the features of his invention. ActiveX uses a Component Object Model; it uses components. *Id.* (quoting interview with ActiveX product managers from Microsoft).

Eolas says this shows that Doyle did not disavow components, in fact he acknowledged them by referencing ActiveX which does use DLLs. Microsoft says that this reference to ActiveX is a self-serving accusation of infringement that should not be read to broaden the scope of the claim. Whether ActiveX employs the '906 invention or not, the point is that Doyle distinguished Khoyi, not by disavowing components as executable applications, but by arguing to the examiner that the '906 method was a different approach.

The inventors did not disavow components as executable applications in order to overcome the examiner's objection with regard to Khoyi.

b. The Koppolu Patent

The third time the examiner rejected the invention, he said it was an obvious combination of the teachings of disclosed prior art (Mosaic, HTTP, HTML, and the World Wide Web) and U.S. Patent No. 5,581,686 (issued Dec. 3, 1996) (the Koppolu Patent). Koppolu teaches a

computer method for "in-place interaction" or "activation in place." The invention allows a user to interact with embedded or linked data in a windowing environment that results from a merger of sorts between a container application and another application, called a server or containee application. Koppolu Patent, col. 7, ll. 3-9. The key to the Koppolu method is OLE (object linking and embedding, pronounced olé). The OLE Application Programming Interface (API) allows applications to communicate with each other by providing a set of functions for container applications to send and receive messages and data to and from server applications. OLE API uses object handlers, dynamically linked code that provides functionality on behalf of the server application. For example, the print function of a spreadsheet program could be written as an object handler. When a spreadsheet object (Koppolu's contained object) is embedded in a word processor document (the compound document), OLE API allows the word processor (the container application) to invoke the spreadsheet application's (the server application's) print function without involving the entirety of the server application's process. *See* Koppolu Patent, col. 9, ll. 11-28. This is an example of the resource-saving capability of dynamically linked components.

In the examiner's view, the Koppolu compound document could be a hypermedia document, and the method taught by Koppolu allowed for the automatic display and interaction of a linked object within a portion of a window controlled by a container application, which could be a web browser. File History, Paper # 15, p. 3. In other words, the examiner thought

that it would have been obvious to combine Koppolu's container method with Web technologies.[13]

The inventors responded with two arguments relevant here. First, Koppolu did not teach the *automatic* invocation of an external function. File History, Paper # 19, pp. 10-11. Second, Koppolu-OLE's object handlers did not allow for the editing of data that the '906 invention specifically sought to accomplish. File History, Paper # 19, pp. 13-14.

Microsoft says that in making these arguments, the inventors disavowed DLLs as executable applications. As with its reading of the Khoyi issue, Microsoft divorces the inventors' arguments from the context in which they were made. First, the patentees did not say that their invention did not use DLLs. Nor did they say their invention used DLLs. Instead, they said that Koppolu's DLLs did not do the same thing as their invention. Quoting from Kraig Brockschmidt's *Inside OLE 2.0* (Microsoft Press 1993), the inventors emphasized that "only when the object is activated does it transition to the running state where the user may perform any number of actions on that object, such as playing or editing the data." File History, Paper # 19, pp. 11-12. The point of this argument was simply to say that Koppolu-OLE did not automatically run the server application, some intermediate user command was required.

The inventors distinguished DLLs as used by Koppolu. Quoting Brockschmidt again, the inventors said "object handlers do not, however, provide any sort of editing facilities for the object itself." File History, Paper # 19, p. 13. The inventors argued that this meant that Koppolu's object handlers did not allow for the "interactive processing" contemplated by their

[13] The Koppolu Patent discusses a preferred embodiment wherein the container application is a word processor and the containee application (or server application) is a spreadsheet program. The preferred embodiment is a means of embedding a spreadsheet object within a word processor document.

invention. To edit the object in the Koppolu system, the inventors argued that the entire server application would have to be implemented; this could be done as an "in-process" DLL. However, to use an in-process DLL to edit object data, the user would have to activate the object (or tell the container application that she wanted to edit the object). File History, Paper # 19, p. 13; *see also* Koppolu Patent, col. 7, ll. 59-63 ("When the user indicates that the budgeting data is to be edited, the word processing program determines which application should be used to edit the budgeting data (e.g., the spreadsheet program) and launches (starts up) that application."). According to the inventors, in-process DLLs did not teach automatic interactive processing of data.

The inventors primarily quoted Brockschmidt for two propositions. One, object handlers could not edit data. Two, in-process DLLs (in-process servers) could not be automatically invoked by Koppolu. Neither of these arguments say that the '906 invention cannot use DLLs; they argue instead that the invention is different than the use of DLLs disclosed in Koppolu.

The inventors did not limit their use of Brockschmidt to these two points. Instead, they noted that Brockschmidt identified other problems associated with object handlers and in-process servers. These problems were: limited cross-platform interoperability and inability to access, or link, to data objects external to the container document. In-process servers, said the inventors, could not *ever* run "stand-alone," so they could never provide linked objects. File History, Paper # 19, pp. 13-14. However, I do not read this argument to suggest that the '906 executable application *must* be standalone or that the inventors denigrated all forms of componentization.

Context is important. The inventors were trying to persuade the examiner that Koppolu OLE was different, it could not do what the '906 invention could do. They may have been artful in their presentation to the examiner. Microsoft attempts to introduce other portions of Brockschmidt, not quoted in the File History, to show that Koppolu OLE DLLs can do anything and everything; therefore, the inventors could only have disclaimed all DLLs to overcome the obviousness objection. But the inventors did not tell the examiner this, they did not quote all of Brockschmidt; instead they said Koppolu DLLs were of limited usefulness. I am not here concerned with the accuracy of the inventors' position, but solely with the position itself.

I find that the inventors, in responding to both the Khoyi and Koppolu patents, did not disclaim or disavow componentization as a programming technique employable as the executable application in the '906 invention.

c. Reasons for Allowance

A separate question is whether the examiner believed the inventors were disclaiming all componentization. After considering Paper # 19, the examiner allowed the patent and said, "the examiner agrees that the claimed executable application is not a code library extension nor object handler (e.g. windows dll and OLE) as pointed out in applicant's argument (paper # 19 pages 12-14)." File History, Paper # 23, p. 3.

Eolas says that the reason for allowance simply says that the examiner agreed with the inventors' argument. Since the argument did not disavow DLLs, the examiner did not exclude DLLs from the scope of the claims. Microsoft says the examiner said what he meant -- the executable application is not a DLL. The Federal Circuit has noted that where an examiner's

reasons for allowance suggest a disavowal of a specific claim construction, and where the applicants fail to respond to the examiner's statement, the patentee's argument that no such disavowal occurred is "particularly unpersuasive." *Elkay*, 192 F.3d at 979.

The examiner's reference to "windows dll" is explicitly tied to Paper # 19. Therefore, the only explanation as to why the examiner believed the executable application is not a "windows dll" must be in Paper # 19. Unless I read the examiner to misunderstand the applicants' argument, the only explanation for the allowance is that the examiner agreed that Koppolu OLE DLLs did not have the functionality of the executable application in the invention. This was the argument made to the examiner. "Windows dll" as used in Paper # 19 means object handlers and in-process servers that cannot edit objects or be automatically invoked. If the basis for the applicants' argument is false, the File History does not reflect that fact, and there is no evidence that the examiner had some broader understanding of the nature of OLE or DLLs in mind when he allowed the patent. Therefore, I read the examiner's reason as an acceptance of the narrow argument offered by the inventors in Paper # 19.

The File History does not limit the term executable application to "standalone" applications, nor does the File History disavow components as a form of executable application. Therefore, I see no need to add any such modifiers to the broad claim term as used by the inventors. An executable application, as used in the '906 Patent, is any computer program code, that is not the operating system or a utility, that is launched to enable an end-user to directly interact with data.

C. "Type Information"

The invention parses a hypermedia document and learns the location of at least a portion of some object. '906 Patent, col. 18, ll. 13-18. This object has "type information" associated with it, and this type information is utilized by the browser to identify and locate the executable application. '906 Patent, col. 18, ll. 18-22. What does type information mean?

1. The Claim and Specification

The claim language suggests that type information provides a clue to the browser to assist it in identifying and locating the executable application. Microsoft seeks to exclude from the scope of "type information" any tag that simply tells the browser which application to use. In its view, type information is limited to data types. For example, according to Microsoft, type information cannot be "WinAmp," it must be ".mp3." If the type information tells the browser what application to use, then the browser has very little left to do in identifying and locating the application. But nothing in the claim language says there has to be a challenge for the browser; if the author of the hypermedia document being parsed wants to make it easy for the browser, and tell it what application to use, so be it. Identifying an application will often convey information as to the type of object involved. For example, identifying the application x-vis conveys that the object is a three-dimensional image; this gives the browser some idea of the character of the object. The claim says type information is associated with the object -- both application names and data types can be associated with objects and both can convey useful information to the browser for it to use in identifying and locating the executable application. Neither possibility is foreclosed by the claim language.

The specification squarely supports this view. The inventors gave examples of type information in the form of the HTML TYPE element of an EMBED tag: "Examples of values

for the TYPE element are 'application/x-vis' or 'video/mpeg'. The type 'application/x-vis' indicated that an application named 'x-vis' is to be used to handle the object..." '906 Patent, col. 13, ll. 2-5. Thus, type information could be either the application itself (x-vis) or the data type (video/mpeg).

Microsoft argues that to simply identify an application no longer "associates" type information with the object and thus would read "associated with" out of the claim language. I disagree. There is no evidence that association is a term of art, and I give it a plain and ordinary meaning. Association requires only some connection between the object and the type information; type information does not have to be integrated into the object to be associated with it. In the example used in the specification, by identifying x-vis as the TYPE element of the EMBED tag, the hypermedia document is associating x-vis with the object, which is identified by the HREF element, also within the EMBED tag.[14] '906 Patent, col. 12, l. 54 - col. 13, l. 18. That is all the claim language requires.

2. The File History

Microsoft says the embodiment in the specification is wrong; the inventors disavowed such a construction of type information in the course of prosecuting the patent. Microsoft acknowledges that this reading would mean that the inventors did not claim one of the embodiments in the specification. It is plausible that during a lengthy prosecution history, patentees fail to update the specification language as they amend the claims. However, in this case, as noted above, I do not find the specification's example of x-vis as the TYPE element to contradict the ordinary meaning of the claim language. *Cf. Novo Nordisk of North America, Inc.*

[14] *See* note 3, *supra*.

v. Genentech, Inc., 77 F.3d 1364, 1369 (Fed. Cir. 1996) (claim language "unquestionably" does not cover specification language). Moreover, given that the primary example used by the patentees in describing the invention, indeed the invention's apparent origin, is the ability to automatically invoke x-vis to allow embedded interaction with 3-D embryo images, I find it difficult to read that embodiment out of the claim. *See Vitronics*, 90 F.3d at 1583 (interpreting claim to exclude preferred embodiment is rarely correct). To reach such a conclusion, the file history must be "highly persuasive." *Id.*

a. The Khoyi Patent

At the time of the examiner's rejection based on Khoyi, the claim language said (emphasis added):

> wherein said first distributed hypermedia document includes an embed text format **that specifies the location of an object** external to the first distributed hypermedia document **and that specifies type information** utilized by said browser to identify and locate an executable application....

The examiner found that Khoyi teaches the ability to "invoke a corresponding object manager (a program external to the document) in response to an invocation request to process and control the object" and teaches "links specifying the object and *type*." File History, Paper # 12, p. 3 (emphasis added). The applicants amended the claim to read (emphasis added):

> wherein said first distributed hypermedia document includes an embed text format... **that specifies the location of at least a portion of an object** external to the first distributed hypermedia document, **wherein said object has type information associated with it** utilized by said browser....

File History, Paper # 14, p. 2. Microsoft says this amendment, by adding "associated with" overcomes the Khoyi prior art by implying that type information must be data type. There is

no substantive discussion in the File History (from either the applicants or the examiner) relating this amendment to Khoyi.[15] There is no discussion of how this amendment changes, if at all, the meaning of type information. Given the tenor of Paper # 14, which notes the limitations of Khoyi, one could read the amendment to be an attempt to broaden, not narrow, the claim. Perhaps the amendment signals to the examiner that this invention can do more than Khoyi – it does not use type information in a Khoyi-like manner specified by the source document, but is more flexible. In any event, given the ambiguous nature of the amendment's relationship to Khoyi, I think it insufficient to mandate a claim construction that excludes a preferred embodiment stated in the specification.

b. The Koppolu Patent

Microsoft next argues that the applicants, in their response to the Koppolu-based objection, revealed an intent to disavow type information as a simple application identifier. Koppolu-OLE, according to the applicants, used a binary pointer mechanism and an operating system registry to identify objects with containee server applications. File History, Paper # 19, p. 9. This CLASSID system, not the compound document's text, is used to determine object type.[16] *Id.* I do not read this to be an explicit disavowal of the possibility that the '906 browser reads a named application as a type associated with an object; instead I read this reference to

---

[15] Professor Felten testified that, as he read the file history, the applicants never distinguished Khoyi based on how type information was used. Felten Direct Examination, October 25, 2000, Tr. at 102. Professor Dunsmore testified that he was not sure why the applicants made the amendment, but that he assumed the change would help in acceptance by the examiner. Dunsmore Direct Examination, October 26, 2000, Tr. at 233.

[16] A CLASSID is a 32-bit number that is a unique identifier for a particular component. Michael Wallent Direct Examination, October 25, 2000, Tr. at 169. The Windows operating system maintains a database registry of CLASSIDs, linking the 32-bit numbers with their corresponding application. Without recourse to this registry, the CLASSID is essentially meaningless.

distinguish a method of using numerical identifiers and platform-dependent registries to perform the association. *See* II.D., *infra.*

Similarly, the applicants' discussion of the cross-platform benefits of the '906 invention does not disavow the possibility that applications are identified as type information. *See* File History, Paper # 19, pp. 21, 25. The applicants were not addressing type information, they were saying their invention is better than OLE because it is not platform-dependent; the invention vests more functionality in the hypermedia document and browser than allowed in the OLE system, according to the applicants. *Id.* Microsoft says it is a logical inference that, since OLE uses the CLASSID architecture, and since that platform-dependent system is generally equivalent to simply naming an application to use with certain objects, the inventors must have been disclaiming this more general approach. I disagree.

In both Paper Nos. 14 and 19, the applicants addressed objections based on prior art, but there is no explicit discussion of how the type information language overcomes the objection. While Microsoft makes plausible arguments by drawing inferences from the file history, I cannot say the history is anything but ambiguous. Given that the claim language supports a construction of type information that includes naming an application, and given that the specification's preferred embodiment explicitly embraces such a form of type information, I reject a reading of the claim that hoists ambiguous file history above the claim and specification.

D. "Utilized By Said Browser To Identify and Locate"

What is the browser supposed to do once it knows of some type information associated with the object? The claim says the type information is "utilized by said browser to identify and locate an executable application external to the first distributed hypermedia document." '906

Patent, col. 18, ll. 20-23. Microsoft asks that I clarify the meaning of this phrase to ensure that it is the browser, not the operating system, that does the "heavy lifting" of utilizing, identifying and locating.

1. The Claim

The claim certainly says that it is the browser, and not any other code, that utilizes the type information to identify and locate the executable application. I read the claim language to mean that the browser identifies and locates the executable application and that it is able to perform these functions because it is armed with the knowledge of type information. Professor Felten agreed that the browser must do the identifying and locating. Felten Cross-Examination, October 25, 2000, Tr. at 149. The question is whether the browser can delegate this function to an outside resource. In other words, can the browser ask the operating system, or perhaps some shared utility, to help it identify and locate an executable application?

Persons skilled in the art define code by its function. Professor Dunsmore, for example, testified that he thinks of browsers and operating systems as defined by functionality. Dunsmore Cross Examination, October 26, 2000, Tr. at 256. Sometimes it is easy to draw a line between browsers and other pieces of code. Functions that are unique to a web browser (*e.g.,* understanding HTML) are considered a part of the browser. *Id.* However, componentization allows for shared functionality -- the lines begin to blur. When asked to identify when a shared function could be considered part of the browser, Professor Dunsmore said it was a tough question. He hypothesized three possibilities: 1) code is part of the browser; 2) the browser invokes some component; or 3) the operating system is asked to perform the function for the browser. *Id.* at 257. Professor Felten also acknowledged the difficulty in articulating the

minimum amount the browser must do for it to be characterized as performing the functions of identifying and locating. Felten Cross-Examination, October 25, 2000, Tr. at 149. However, I believe that experts are able to make such a judgment when presented with specific code.

The claim language assigns the functions of identifying and locating the executable application to the browser. Whether the browser is performing these functions in any given permutation is a question of fact.

2. The Specification

In a preferred embodiment in the specification, the browser, not the operating system, identifies and locates the executable application. However, the browser does not work alone. The specification makes clear that the inventors contemplated the browser's use of some outside resources. Microsoft agrees that operating systems are always involved on some level, and Microsoft also agrees that the specification discloses the use of outside resources. Microsoft does not propose a claim construction that would entirely preclude the browser from using the operating system or some external resource. In the specification, the browser, armed with the type information, consults a user-defined list of application type/application pairs, such as the MIME (Multipurpose Internet Mail Extensions) database. '906 Patent, col. 15, ll. 13-18. The parties agree that the MIME database is external to the browser. Microsoft's position is that this embodiment is consistent with its construction because it is the browser that consults the MIME database, and it is the browser that uses the MIME database to learn the application type. This example is also consistent with Eolas's broad definition of "utilize" – to put to use. The browser puts the type information to use by taking it to some outside resource and then using the

resource to identify and locate the executable application. This is exactly what the claim language says is supposed to happen.

The parties' dispute over this term appears to be more properly viewed as an infringement question than a claim construction issue. One infringement question will be whether Microsoft's browser, Internet Explorer, identifies and locates executable applications. This is not a question I can answer, yet. All I can decide is that the claim language means what it says, the functions must be performed by the browser.

3. The File History

By arguing that both Khoyi and Koppolu-OLE were different because they were operating system-dependent, the inventors highlighted the difference between having the browser link an object type to an application and using OLE's CLASSID to perform that function. In Khoyi, according to the inventors, "the object managers for different data types are coordinated by the operating system so that each type of displayed data is rendered by its associated object manager, the actual linking operations are coordinated by the operating system." File History, Paper # 14, p. 14. In attempting to overcome the Koppolu-based rejection, the inventors said the same thing: "the actual linking mechanism between the container document and the containee server application is coordinated by the operating system's registry database." File History, Paper # 19, p. 9. I read the inventors' argument as saying not just that the operating system maintains a registry that a browser can use, but that in OLE, it is the operating system itself that performs the linking function. This is different than the invention.

The claim language, the specification and the File History all suggest that the functions of using type information to identify and locate the executable application must be performed

by the browser. No one suggests that the browser must do it alone, and I do not construe the claim to require that. However, I accept Microsoft's construction that, as a factual matter, one must be able to characterize the browser as doing the heavy lifting, which is what it does in the specification. Neither the claim nor the specification give adequate guidance as to what heavy lifting may be because neither had a need to address the issue. This is not surprising. "Utilize" is a common English word, and there is no evidence of a particular meaning it may have to those skilled in the art.

This does not mean the question is unanswerable, it merely means the answer lies in specific factual contexts. This is evidenced by both experts' inability to articulate a definition in the abstract. A careful examination of code would be necessary to decide what the browser is utilizing and how it is utilizing it. Framing this as a claim construction question, Eolas took the view that "by said browser" did not exclude some use of the operating system, probably because it viewed Microsoft as contending that the browser must do it all by itself. Microsoft did not take this position. It conceded that the operating system was necessarily a part of the browser's arsenal. Microsoft stood on the proposition that the browser had to do the heavy lifting in contrast to OLE, where the operating system performs the enumerated functions. Beyond this, both sides left the specifics undeveloped. Therefore I am left to simply construe the claim language to mean what it says, the functions of utilizing the type information to identify and locate the executable application must be performed by the browser, not the operating system as in Koppolu's OLE.

III. Conclusion

A. An "executable application," as used in the '906 Patent, is any computer program code, that is not the operating system or a utility, that is launched to enable an end-user to directly interact with data.

B. "Type Information" may include the name of an application associated with the object.

C. "Utilized by said browser to identify and locate" means that the enumerated functions are performed by the browser. This is a fact-intensive inquiry.

Enter opinion and order construing disputed claim terms.

ENTER:

_____
James B. Zagel
United States District Judge

DATE: _____