

# **EXHIBIT D2**

manner designed to provide maximum benefit to members of a collaborative engineering team.

For example, messages are inserted in chronological order as "NoteMail" pages in an approach that departs from the functionality of prior art web browsers. Prior art web browsers typically organize web page retrieval around stored bookmarks that provide URL links to web pages (and associated objects) that may reside anywhere on the Internet.

The Examiner concurs with the Patent Owner's contention that the "NoteMail" design (i.e., teaching restricted, collaborative access to a centralized database) runs counter to the intended purpose of the Mosaic (APA), Berners-Lee, Raggett I and II hypothetical four-way combination. The intended purpose of the four-way combination is to provide a distributed system that enables universal access to web pages (and associated objects) that may be stored anywhere on the Internet.

In contrast, Toye discloses a system that permits applications to reside anywhere on the Internet, while collaborative, restricted access to the data is only permitted via a centralized database [Toye, p. 40, col. 2, paragraph 2, last line].

Accordingly, the Examiner concurs that the Toye reference teaches away from modifying the Mosaic (APA), Berners-Lee, Raggett I and II hypothetical four-way combination as proposed by the rejection set forth in the last office action.

*PART V. The secondary consideration of commercial success further supports the conclusion of non-obviousness. The attached Declaration of Robert J. Dolan, Dean at the University of Michigan Business School and Gilbert and Ruth Whitaker professor at Michigan Business School ("Dolan") sets forth facts and evidence to legally and factually establish the secondary consideration of commercial success of the invention claimed in claims 1 and 6 of the '906 patent.*

*a. There is a nexus between the claimed invention and the commercial success.*

In response to the Patent Owner's argument V(a), the Examiner has reviewed the supporting "Dolan" Declaration and does not find it persuasive in terms of demonstrating a nexus between the instant claimed '906 invention and commercial success.

The "Dolan" Declaration relies upon the alleged infringement of the '906 patent claims by Microsoft in marketing the Microsoft Internet Explorer browser (IE). In particular, it is alleged that the "IE browser's support for "plug-ins, applets, and Active X functionality incorporates the technology claimed in claims 1 and 6 of the '906 patent" [See "Dolan" Declaration, page 2].

The Patent Owner's argument of commercial success is thus predicated on Microsoft's infringement of the '906 patent as determined by a jury in the trial at the U.S. District Court (Northern District of Illinois, Eastern Division).

However, at the time of this writing, the litigation is still ongoing and is on remand back to the District Court from the CAFC.

While the infringement issue is not being considered on remand from the CAFC, the affirmative defenses of public use and inequitable conduct, if successful, would render the patent invalid and the issue of patent infringement would be moot. Therefore, the PTO does not consider there to be a final judgment on the issue of patent infringement until all appeals have been exhausted and the litigation has concluded. A nexus between the claimed invention and the commercial success of the IE browser cannot be shown (based upon alleged patent infringement) in the absence of a final judgment to establish such infringement.

Accordingly, the Patent Owner has not met the burden of proof required to establish a factual and legally sufficient connection between the evidence of commercial success and the claimed invention such that the evidence is of probative value in the determination of nonobviousness.

*b. The evidence of commercial success is commensurate with the scope of the '906 claims.*

Objective evidence of nonobviousness including commercial success must be commensurate in scope with the claims. In re Tiffin, 448 F.2d 791, 171 USPQ 294 (CCPA 1971). In order to be commensurate in scope with the claims, the commercial success must be due to claimed features, and not due to unclaimed features. Joy Technologies Inc. v. Manbeck, 751 F. Supp.

225, 229, 17 USPQ2d 1257, 1260 (D.D.C. 1990), *aff'd*, 959 F.2d 226, 228, 22 USPQ2d 1153, 1156 (Fed. Cir. 1992).

The Patent Owner relies upon the "Dolan" Declaration (pages 6-9, numbered paragraphs 25-41) to support the contention that the evidence of commercial success is commensurate in scope with claims 1 and 6 of the instant '906 patent.

In response to the Patent Owner's argument V(b), the Examiner need not reach this issue because a nexus between the claimed invention and commercial success has not been established, as discussed in the response to argument V(a), *supra*. A nexus between the claimed invention and the commercial success of the IE browser cannot be shown (based upon alleged patent infringement) in the absence of a final court judgment to establish such infringement (i.e., until all appeals have been exhausted and the litigation has concluded).

The Examiner cannot reasonably address the issue raised by argument V(b) without commenting on the merits of the ongoing litigation. Subject matter concerning patent infringement constitutes a federal question that properly falls within the subject matter jurisdiction of the Federal Court system. Subject matter concerning patent infringement is not considered by the U.S. Patent and Trademark Office.

*c. The commercial success is derived from the invention.*

In response to the Patent Owner's argument V(c), the Examiner does not find the Patent Owner's arguments and the associated "Dolan" Declaration persuasive for the following reasons:

In considering evidence of commercial success, care should be taken to determine that the commercial success alleged is directly derived from the invention claimed, in a marketplace where the consumer is free to choose on the basis of objective principles, and that such success is not the result of heavy promotion or advertising, shift in advertising, consumption by purchasers normally tied to applicant or assignee, or other business events extraneous to the merits of the claimed invention, etc. *In re Mageli*, 470 F.2d 1380, 176 USPQ 305 (CCPA 1973) (conclusory statements or opinions that increased sales were due to the merits of the invention are entitled to little weight); *In re Noznick*, 478 F.2d 1260, 178 USPQ 43 (CCPA 1973).

Even assuming, arguendo, that the Patent Owner has demonstrated the required nexus between the instant claimed '906 invention and the commercial success of an allegedly infringing product, the "Dolan" Declaration fails to show that the commercial success of Microsoft's IE browser was not the result of heavy promotion or advertising or other business events extraneous to the merits of the claimed invention.

In particular, Microsoft made the IE browser available to users at little or no cost. Microsoft also bundled the IE browser as an integral component of various Microsoft operating systems (e.g., Windows 95, 98, and Windows

2000). Significantly, the "Dolan" Declaration is silent regarding the issue of free or low cost distribution of the IE browser as a factor in Microsoft's successful capture of market share.

In more traditional business models that involve tangible products, a rational producer will seek to exploit a profit opportunity until the marginal cost of the  $n^{\text{th}}$  unit produced exceeds the marginal revenue generated from that  $n^{\text{th}}$  unit. However, when software is distributed over the Internet, the marginal cost of each unit of downloaded software approaches zero as the number of downloads approaches infinity. This is true because the sunk software development costs and the relatively fixed cost of maintaining distribution servers are averaged over a potentially infinite number of downloads.

Obviously, if there exists a quantifiable market demand for a given product, the quantity of units demanded will increase as the cost per unit approaches zero. This was likely true in the case of the Microsoft IE browser because it was offered to the public as a free download (or merely for the cost of the CD media plus postage and handling).

Microsoft clearly offered the IE browser to the public at little or no cost in an effort to gain market share over the competing Netscape browser, even though it may also be true that Microsoft viewed the functionality of Active X (allegedly infringing upon the '906 patent functionality) as giving IE an advantage over Netscape [e.g., see "Dolan" Declaration, page 8, paragraph 36]. In addition, such free distribution of the IE browser clearly promoted and helped to advertise Microsoft's main operating system and application software products.

Because Microsoft made the IE browser available to the public at little or no cost, the past distribution of IE has at least the appearance of "heavy promotion or advertising." While the alleged infringement of '906 functionality may indeed have been a factor in the market success of the IE browser, patent infringement has not been shown by a final court judgment. Significantly, the Patent Owner has failed to address the Microsoft marketing strategy of distributing the IE browser to the public at little or no cost.

Because Microsoft was already an established market leader with respect to desktop operating systems and applications, the success of the IE browser could also be reasonably attributed to Microsoft's extensive advertising and position as a market leader before the introduction of the allegedly infringing product (i.e., the IE browser). See Pentec, Inc. v. Graphic Controls Corp., 776 F.2d 309, 227 USPQ 766 (Fed. Cir. 1985) (commercial success may have been attributable to extensive advertising and position as a market leader before the introduction of the patented product).

Accordingly, even when the facts are viewed in a light most favorable to the Patent Owner, the Patent Owner has failed to demonstrate by a preponderance of the evidence that the commercial success of Microsoft's IE browser was derived from the instant '906 invention.



**The Viola Code**

The CAFC opinion (Docket No. 04-1234, March 2, 2005) states on page 11, 2<sup>nd</sup> paragraph:

In contrast, the record indicates Wei not only demonstrated DX34 to two Sun Microsystems engineers without a confidentiality agreement (on May 7, 1993), but only twenty-four days later (on May 31, 1993) posted DX37 on a publicly-accessible Internet site and notified a Sun Microsystems engineer that DX37 was available for downloading.

The '906 invention was reduced to practice no later than January, 27, 1994 when it was presented on that date to a conference "Medicine Meets Virtual Reality II."<sup>2</sup> From the court record, it is clear that the date of publication on the Internet of the DX37 code (May 31, 1993) antedates the date of reduction to practice (Jan. 27, 1994) of the '906 invention. Accordingly, the DX37 code submitted by the Patent Owner on Dec. 30, 2003 (received by the PTO on Jan 5, 2004) has been considered by the Patent and Trademark Office as a publication that constitutes prior art for purposes of this reexamination proceeding.

The "Viola Code" is stored as an artifact (i.e., a CD disk) associated with the instant Image File Wrapper (IFW) reexamination file. The contents of artifacts are not stored as images on the PTO IFW system. The Viola code CD contains two compressed zip files representing "Viola Source code" ("DX34" and "DX37");

---

<sup>2</sup> See "Ruling on the Defense of Inequitable Conduct", No. 99 C 626, U.S. District Court Northern District of Illinois, Eastern Division, page 9.

- 1) viola930512.tar.gz.zip - this compressed file represents the earlier Viola source code, also referred to as "DX34" in the CAFC opinion (Docket no. 04-1234, March 2, 2005, see also IFW "Reexam Notice of Court Action" dated April 11, 2005; see especially page 11 as numbered in the printout (corresponding to IFW page 16 of 32). The viola930512.tar.gz.zip (i.e., "DX34") file, when unzipped, contains 1,027 files in 35 folders consisting of 8 total megabytes in size.
- 2) violaTOGO.tar.Z.zip - this compressed file represents the later Viola source code, also referred to as "DX37" in the CAFC opinion. The violaTOGO.tar.Z.zip (i.e., "DX37") file, when unzipped, contains 1,030 files in 34 folders consisting of 7.7 total megabytes in size.

To conduct a thorough and comprehensive review of the DX37 code (1,030 files), the Examiner successfully unzipped the provided "violaTOGO.tar.Z.zip" compressed file and indexed all DX37 files using a commercially available text searching program designed for such purpose.<sup>3</sup>

In this manner, every DX37 file containing textual content (including code) was fully and comprehensively text searched with the resulting "hits" being highlighted in the full-text context of each document. Several representative Viola files are reproduced *infra* to clarify the scope of the Viola DX37 prior art publication.

---

<sup>3</sup> The Examiner used the "dtSearch" program to index and text search all DX7 files that contained textual content. See <http://www.dtsearch.com/>

### **How Viola embeds Viola scripts in a hypermedia document**

In particular, the file "violaApps.hmml" (contained in the "docs" directory) illustrates how interactive applications (i.e., actually Viola scripts) are embedded in a Viola hypermedia document as designated by a matched pair of <VOBJF> and </VOBJF> tags that specify a Viola script that is used to generate the embedded object, as shown below:

```
<VOBJF> ../apps/clock.v </VOBJF>
```

When the Viola hypermedia browser parses the hypermedia document (e.g., "violaApps.hmml", denoting a hypermedia document written in Hyper Media Markup Language) and encounters the matched pair <VOBJF> and </VOBJF> tags, the browser then retrieves the Viola script "clock.v" from the directory location specified by the directory path (i.e., ../apps/ ).

Significantly, the Viola script "clock.v" is INTERPRETED to embed an interactive application object within the same window of the Viola browser. Each Viola script line is interpreted by translating the Viola script code (or corresponding byte code) to native binary machine code instructions that are executed in a sequential fashion.

The Viola documentation states: "The extension language is C-like in syntax and is processed into byte-code for efficient interpretation" [see "violaCh1.hmml" in the "docs" directory]. Although the aforementioned "clock.v" example is clearly a Viola script, it appears that an intermediate byte-code representation may be interpreted at runtime. In such case, the Viola script must be compiled in advance to intermediate byte-code form.

The "violaApps.hmml" hypermedia document file (as parsed by the Viola browser) and the corresponding "clock.v" script file are shown below:

"violaApps.hmml" illustrating the use of the Viola <VOBJF> object tags (located within the "docs" directory)

```
<!DOCTYPE hmml SYSTEM>
<TITLE>Test</TITLE>
<H1>List No. 5</H1>
<P>
The <CMD>&lt;VOBJF&gt;</CMD> tag can be used to insert viola
applications.
Using this capability allows you embed in your document what you
can access or build using viola's programming, and GUIs. Of
course too much violaism reduces the portability of your document
on the World Wide Web,but anyway...
<P>
Here are some examples.
<H2>Clock</H2>
<VOBJF>../apps/clock.v</VOBJF>
<H2>Vicon</H2>
<VOBJF>../apps/vicon.v</VOBJF>
<P>
This can be a handy menu to tuck away at a corner of the screen.
<H2>Query</H2>
<VOBJF>../apps/vwq.v</VOBJF>
<P>
This application is intended to gather user information.
<H2>Wave fun</H2>
<VOBJF>../apps/wave.v</VOBJF>
<H2>Noodle Doodles</H2>
<VOBJF>../apps/doodle.v</VOBJF>
<P>
So I was bored...
<P>
The end.
```

The first portion of the corresponding "clock.v" Viola script (located in the "apps" directory)

```
\name {clock}
\class {vpane}
\parent {}
\width {200}
\height {210}
\children {clock.dial clock.mesg}
```

```
\
\name {clock.dial}
\class {XPMBG}
\parent {clock}
\script {
  print("@@@@@ clock: ");
  for (i =0; i < arg[]; i++) print(arg[i], ", ");
  print("\n");

  switch (arg[0]) {
  case "tick":

    date = date();
    clock.mesg("update");
    second = int(nthWord(date, 6));
    minute = int(nthWord(date, 5));
    hour = int(nthWord(date, 4));
    if (hour >= 12) hour = hour - 12;

    secondD = (second / 60.0 * 360.0) - 90.0;
    minuteD = (minute / 60.0 * 360.0) - 90.0;
    hourD = (hour / 12.0 * 360.0) - 90.0 + (minute / 60.0 * 30.0);

    secondX = secondR * cos(secondD) + centerX;
    secondY = secondR * sin(secondD) + centerY;
    minuteX = minuteR * cos(minuteD) + centerX;
    minuteY = minuteR * sin(minuteD) + centerY;
    hourX = hourR * cos(hourD) + centerX;
    hourY = hourR * sin(hourD) + centerY;

    if (lminuteX != minuteX) {
      clearWindow();
      clock.dial("render"); /* brutally redraw */
      drawLine(centerX, centerY, minuteX, minuteY);
      drawLine(centerX, centerY, hourX, hourY);
      invertLine(centerX, centerY, lsecondX, lsecondY);
    }
    invertLine(centerX, centerY, lsecondX, lsecondY);
    invertLine(centerX, centerY, secondX, secondY);

    lsecondX = secondX;
    lsecondY = secondY;
    lminuteX = minuteX;
    lminuteY = minuteY;
    lhourX = hourX;
    lhourY = hourY;
    if (view) after(1000, "clock.dial", "tick");
    return;
  break;
  case "render":
    usual();
    for (i = 1; i <= 12; i = i + 1) {
```

```
        x = letterR * cos((i / 12.0 * 360) - 90) +
            centerX - 10;
        y = letterR * sin((i / 12.0 * 360) - 90) +
            centerY - 5;
        drawText(x, y, i, str(i));
    }
    return;
break;
case "VIEW_ON":
    view = 1;
    return;
break;
case "VIEW_OFF":
    view = 0;
    return;
break;
case "expose":
    clearWindow();
    lminuteX = 0; /* forces redrawing */
    lhourX = 0; /* forces redrawing */
break;
case "config":
    usual();
    send(self(), "resize", arg[3], arg[4]);
    return;
break;
case "resize":
    if (arg[1] < arg[2])
        radius = arg[1] / 2.0;
    else
        radius = arg[2] / 2.0;

    centerX = arg[1] / 2.0;
    centerY = arg[2] / 2.0;
    secondR = radius * 0.95;
    minuteR = radius * 0.9;
    hourR    = radius * 0.6;
    letterR = (radius - 9) * 0.94;

    after(2000, "clock.dial", "tick");
    lminuteX = 0;

/*      system(concat(environVar("VIOLA"), "/play ",
*/              environVar("VIOLA_DOCS"), "/cuckoo.au"));
break;
}
usual();
}
```

The Viola DX37 approach to embedding interactive objects using interpreted Viola scripts (or corresponding byte-code forms) does not anticipate nor fairly suggest the '906 invention as claimed for at least the following reasons:

While Viola DX37 supports hypermedia and a type of interpreted script-based interactive processing, the Examiner can find no indication from a comprehensive text search of the Viola DX37 files that such interactivity results from the use of a parsed **embed text format** that specifies the location of an **object** external to the hypermedia document, where the browser application **uses type information associated with the object to identify and locate an external executable application**, and where the parsing step results in the browser automatically invoking the **executable application** to display the **object** and enable interactive processing of the **object** within the same browser-controlled window, when the instant '906 patent claims 1 and 6 are properly accorded the broadest reasonable interpretation consistent with the specification.

**I. VIOLA <VOBJF> TAGS DO NOT  
ANTICIPATE NOR FAIRLY SUGGEST THE  
"EMBED TEXT FORMAT" AS CLAIMED IN  
THE '906 PATENT.**

Unlike the instant '906 claimed "embed text format," the Viola <VOBJF> tags use no arguments or additional elements beyond a directory path and filename. The Viola <VOBJF> tag simply loads the Viola script using the

path and filename specified between the <VOBJF> and </VOBJF> tags, as shown:

```
<VOBJF> ../apps/clock.v </VOBJF>
```

In contrast, the browser application of the instant '906 patent uses a type element associated with the external object (i.e., "type information" as claimed) to identify and locate an executable application external to the distributed hypermedia document [see '906 patent, TABLE II and associated discussion col. 13].

Significantly, the Viola browser application does not fairly teach nor suggest where the browser application uses type information associated with the external object to identify and locate an external executable application.

**II. VIOLA SCRIPTS (OR CORRESPONDING BYTE-CODE FORMS) DO NOT ANTICIPATE NOR FAIRLY SUGGEST THE EXTERNAL "OBJECT" AS CLAIMED IN THE '906 PATENT.**

If the Viola <VOBJF> tags are considered as arguably corresponding to the instant claimed '906 "embed text format" (in the sense that the Viola <VOBJF> tags specify "the location of at least a portion of an object external to the first distributed hypermedia document" as claimed in '906 claims 1 and 6), then the Viola script program specified between the <VOBJF> tags is not equivalent to the instant '906 claimed external "object" when the



claimed '906 external "object" is interpreted in a manner consistent with the specification of the '906 patent.

The Viola, "clock.v" script is a high-level source code PROGRAM. In contrast, the scope of the claimed '906 external "object" broadly encompasses myriad types of data objects, including self-extracting data objects [see '906 patent, col. 3, lines 33-51].

The scope of the claimed '906 external "object" is broad when construed in a manner consistent with the specification (i.e., see '906 patent, col. 3, lines 36-39: "a data object is information capable of being retrieved and presented to a user of a computer system."). However, the scope of the claimed '906 external "object" clearly does not read upon a high-level source code PROGRAM, such as a Viola script, nor does it read upon an object in byte-code form.

When the scope of the claimed '906 external "object" is construed in a manner consistent with the specification, it is clear that any executable component of the claimed '906 external data "object" is limited to performing self-extraction of the compressed data object:

See '906 patent, col. 3, lines 43-51:

When a browser retrieves an object such as a self-extracting **data object** the browser may allow the user to "launch" the self-extracting **data object** to automatically execute the unpacking instructions to expand the data object to its original size. Such a combination of executable code and data is limited in that the user can do no more than invoke the code to perform a singular

function such as performing the self-extraction after which time the object is a standard data object.

Although a self-extracting data object typically includes executable code to expand the compressed data object to its original size, this type of self-extraction extracts DATA that has no relationship to a high-level source code PROGRAM in the form of a Viola script, or a byte-code file, or the like.

**III. VIOLA SCRIPTS (OR CORRESPONDING  
BYTE-CODE FORMS) DO NOT ANTICIPATE  
NOR FAIRLY SUGGEST THE EXTERNAL  
"EXECUTABLE APPLICATION" AS CLAIMED  
IN THE '906 PATENT.**

The Examiner finds that the Viola code publication does not fairly teach nor suggest that the browser automatically invokes an **executable application**, external to the hypermedia document, to display the object and enable interactive processing of the object, when the instant '906 patent claims 1 and 6 are properly accorded the broadest reasonable interpretation consistent with the specification, where such interpretation is also consistent with the interpretation that those skilled in the art would reach. In re Hyatt, 211 F.3d 1367, 1372, 54 USPQ2d 1664, 1667 (Fed. Cir. 2000); In re Cortright, 165 F.3d 1353, 1359, 49 USPQ2d 1464, 1468 (Fed. Cir. 1999).

While expert witnesses and dictionaries (considered as extrinsic evidence) may differ regarding the proper construction of the instant claimed "executable application", the Central Processing Unit (i.e., CPU or

microprocessor) found in every computer system has only a single, precisely defined interpretation as to what constitutes an "executable application."

When the CPU initiates a "fetch and execute" cycle, the program counter is loaded with the address of the next executable instruction. To be "executable" the contents of the memory location pointed to by the program counter must contain an instruction in binary form that is a member of the native instruction set of the microprocessor (i.e., a binary machine language instruction). The binary representation of the precise portion of the machine language instruction that determines what kind of action the computer should take (e.g., add, jump, load, store) is referred to as an operation code (i.e., OP code). From the perspective of the CPU, if a recognizable machine language instruction (i.e., a native CPU instruction) is not found within the memory location pointed to by the program counter, the computer will crash.

The Viola system uses "C-like" Viola scripts that must be INTERPRETED by the browser and then TRANSLATED or CONVERTED into binary native executable machine code that can be understood by the CPU. Alternately, the Viola script is precompiled to intermediate byte-code form and the byte-code is interpreted (i.e., translated) into binary native executable machine code at runtime. This extra step of translation results in an unavoidable performance penalty, as interpreted applications run much slower than compiled native binary executable applications.

Accordingly, the "C-like" Viola scripts (or corresponding byte-code representations) are not "executable applications" from the perspective of the CPU, which is the only perspective that really matters at runtime. A

conventional CPU is only capable of processing binary machine language instructions from its own native instruction set.

Without an intermediate translation step performed by an interpreter component of the Viola browser, a Viola script (or corresponding byte-code representation) cannot be processed as an executable application by the CPU.

Significantly, the instant '906 specification is silent regarding the use of applications that rely upon scripts that must be interpreted before they can be executed. The instant '906 specification is silent with respect to interpreting code prior to execution. The instant '906 specification is silent with respect to the use of byte-code intermediate forms.

**IV. THE INTENDED USE OF THE VIOLA RAPID  
PROTOTYPING INTERPRETED SCRIPTING  
SYSTEM TEACHES AWAY FROM THE  
INTENDED USE OF THE '906 PATENT.**

The Viola scripting system teaches away from the primary intended use of the '906 invention. The main object of the Viola scripting system was to provide an interpreted operating environment primarily designed for rapid prototyping.

In contrast, the main object of the '906 invention is to provide a system "that allows the accessing, display and manipulation of large amounts of

data, especially image data, over the Internet to a small, and relatively cheap, client computer ['906 patent, col. 6, lines 21-25].

The use of an interpreted script application (or corresponding intermediate byte-code representation) in the '906 patent context would be unacceptably slow in processing large amounts of data, especially the kind of complex three-dimensional image data used in one embodiment of the '906 patent. One must reflect on the fact that the personal computers used in 1994 were significantly slower than the high speed computers widely used today (2005).

Overcoming the existing bandwidth and processing speed constraints associated with the prior art are central objects of the '906 invention [see '906 patent, col. 5, lines 39-56]:

The open distributed hypermedia system provided by the Internet allows users to easily access and retrieve different data objects located in remote geographic locations on the Internet. However, this open distributed hypermedia system as it currently exists **has shortcomings** in that today's large data objects are limited largely by **bandwidth constraints** in the various communication links in the Internet and localized networks, and by the limited processing power, or computing constraints, of small computer systems normally provided to most users. **Large data objects are difficult to update at frame rates fast enough (e.g., 30 frames per second) to achieve smooth animation. Moreover, the processing power needed to perform the calculations to animate such images in real time does not exist on most workstations, not to mention personal computers. 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.**

Also see '906 patent, col. 6, lines 21-31:

On the other hand, small client computers in the form of personal computers or workstations such as client computer 108 of FIG. 2 are generally available to a much larger number of researchers. Further, it is common for these smaller computers to be connected to the Internet. **Thus, it is desirable to have a system that allows the accessing, display and manipulation of large amounts of data, especially**

**image data, over the Internet to a small, and relatively cheap, client computer.**

Due to the relatively **low bandwidth of the Internet** (as compared to today's large data objects) and the **relatively small amount of processing power available at client computers**, many valuable tasks performed by computers cannot be performed by users at client computers on the Internet.

The importance of "speed of access" to application client 210 (corresponding to the instant claimed "executable application") is further demonstrated by the use of "Terminate and Stay Resident" (TSR) programs to provide faster access [See '906 patent, col. 8, lines 66, 67, cont'd, col. 9, lines 1-14]:

Client computer 200 includes processes, such as browser client 208 and **application client 210**. In a preferred embodiment, **application client 210** is resident within client computer 200 prior to browser client 208's parsing of a hypermedia document as discussed below. In a preferred embodiment application client 210 resides on the hard disk or RAM of client computer 200 and is loaded (if necessary) and executed when browser client 208 detects a link to **application client 210**. The preferred embodiment uses the XEvent interprocess communication protocol to exchange information between browser client 208 and **application client 210** as described in more detail, below. Another possibility is to install **application client 210** as a "**terminate and stay resident**" (TSR) program in an operating system environment, such as X-Window. Thereby making access to **application client 210** much faster.

The Examiner submits that "Terminate and Stay Resident" (TSR) programs were notoriously understood to be native binary executable code by those of ordinary skill in the art at the time of the '906 invention. <sup>4</sup>

For example, in the legacy Microsoft MS-DOS environment, TSR programs were native binary executables designated as COM or EXE programs that were preloaded in memory for fast execution. TSR programs were typically used to allow utilities, drivers, or interrupt handlers to be preloaded in

memory for quick access.<sup>5</sup> The purpose of memory preloading for quick access would not be well served if a TSR program in the form of a script had to be interpreted (i.e., translated) to binary native code before it could be executed.

In addition, the '906 patent teaches the use of applications such as "spreadsheet programs, database programs, and word processor programs" [see col. 13, line 14]. The Examiner submits that at the time of the invention most commercial spreadsheet programs, database programs, and word processor applications were usually sold as native binary executable applications. The Examiner does concede that applications of the aforementioned types were available in interpreted languages at the time of the invention (e.g., a database program written in the BASIC language). However, an interpreted application in source code form cannot be executed directly by the CPU without first being translated to native binary executable machine code form, as discussed *supra*.

---

<sup>4</sup> See e.g., U.S. Patent 5,056,057 to Johnson et al., "Keyboard interface for use in computers incorporating **terminate-and-stay-resident** programs", issued Oct. 8, 1991.

<sup>5</sup> Duncan, Ray, "Advanced MSDOS Programming", Microsoft Press, 1986, page 391.

- V. **EVEN ASSUMING, ARGUENDO, THAT "INTERPRETING A SCRIPT" (OR CORRESPONDING BYTE-CODE REPRESENTATION) MAY BE BROADLY CONSIDERED AS EQUIVALENT TO "EXECUTING AN APPLICATION", SUCH INTERPRETATION MERGES THE BROWSER AND THE "EXECUTABLE APPLICATION" INTO ONE PROGRAM THAT FAILS TO TEACH EVERY ELEMENT OF THE '906 PATENT CLAIMS.**

Assuming *arguendo* that one adopts the alternate broader modern construction where "interpreting a script" (or interpreted the corresponding byte-code representation) may be considered as equivalent to "executing an application," then the Viola script arguably becomes an integral component of the Viola browser that parses, interprets (i.e. translates), and executes each line of the script (or corresponding byte-code). In such case, the browser and the "executable application" merge into one program, and therefore cannot meet the requirement for a discrete "browser application" and a discrete "executable application" as claimed by the instant '906 patent [see claims 1 and 6].

Lastly, The Examiner takes particular note of the fourth line of the "violaBrief.hmml" file ("Technical Overview of Viola," see the "docs"



Reexamination/Control Number:  
90/006,831  
Art Unit: 3992

Page 61

directory) that leads one to conclude that the Viola DX37 invention may not have been fully enabled at the time of publication:

<TITLE>Viola, A Technical Summary</TITLE>  
<CAUTION>THIS DOCUMENT IS IN DRAFT STATUS</CAUTION>

For at least the aforementioned reasons, the DX37 Viola files, when considered as a prior art publication for purposes of reexamination, do not teach nor fairly suggest the instant '906 invention, as claimed.

An appendix is attached that presents some of the more relevant Viola documentation files. The files were created for display by a Viola browser and are presented with the included hypermedia tags as found on the CD artifact disk.

EOLASTX-0000001360

**Conclusion**

In summary, the Examiner concurs with the Patent Owner with respect to arguments I-IV for the reasons discussed *supra*.

Although the Examiner does not concur with the Patent Owner with respect to argument V, the issue of establishing a nexus between the claimed invention and commercial success is not dispositive.

The Patent Owner's arguments traversing the rejection need only prevail by the "preponderance of the evidence" standard to succeed in having the rejections set forth in the last office action withdrawn. The ultimate determination of patentability must be based on consideration of the entire record, by a preponderance of evidence, with due consideration to the persuasiveness of any arguments and any secondary evidence. In re Oetiker, 977 F.2d 1443, 24 USPQ2d 1443 (Fed. Cir. 1992).

Accordingly, for at least the aforementioned reasons set forth with respect to arguments I-IV, the Examiner has reconsidered and withdrawn the rejections set forth in the last office action (mailed Aug. 16, 2004).

In addition, the DX37 Viola files have been considered as a prior art publication. For the reasons discussed *supra*, the DX37 Viola files included on the CD artifact do not teach nor fairly suggest the instant '906 invention, as claimed.

Reexamination/Control Number:  
90/006,831  
Art Unit: 3992

Page 63

Instant U.S. Patent 5,838,906 claims 1-10 are hereby confirmed.

Any comments considered necessary by PATENT OWNER regarding the above statement must be submitted promptly to avoid processing delays. Such submission by the patent owner should be labeled: "Comments on Statement of Reasons for Patentability and/or Confirmation" and will be placed in the reexamination file.

St. John Courtenay III  
Primary Examiner  
Central Reexamination Art Unit 3992

Mark Reinhart, First Conferee  
Special Programs Examiner (SPRE)  
Central Reexamination Art Unit 3992

Second Conferee

EOLASTX-0000001362

## VIOLA APPENDIX

The contents of file "viola.desc" (contained in the "docs" directory):

language: viola (Visual Interactive O Language and Application)  
package: viola  
version: 2.0 beta  
parts: interpreter, applications, documentation,  
how to get: ftp xcf.berkeley.edu src/local/viola/\*  
description: A language/toolkit for hypermedia applications. Very loosely modeled after Hypercard. Intended as a tool for building and running hypermedia applications that are composed of collection of classed objects interacting with the user and passing messages among each other. Has simple GUI specification language. Is event driven (X-Window, timer, I/O). Notion of "objects" for modularization and scalability. Syntax is C like. Bytecode compilation is done incremental (object by object). Is single class inheritance.  
+ has objects  
+ dynamic array  
+ message passing  
+ bytecode compiler, interpreter  
+ graphical interface toolkit  
+ pseudo-terminal I/O interface  
+ socket I/O interface  
+ world wide web interface  
- non dynamic class definition (C level definition)  
- non dynamic data types: string, char, int, float, array  
- little interactive authoring tools for naive users  
- development on language is slow, but application driven  
ports: Unix/X  
author: Pei Y. Wei <wei@xcf.berkeley.edu>  
status: actively developed, application driven  
discussion: viola@xcf.berkeley.edu  
updated:  
-----  
reference: The X Resources, O'Reilly & Associates  
europe: ftp info.cern.ch pub/www/src/viola\*  
japan: ftp srawgw.sra.co.jp pub/x11/viola\*

The file "violaBrief.hmml" (contained in the "docs" directory) provides a technical overview of Viola, and is reproduced for the record in its entirety below (including the "hmml tags" associated with the browser hypermedia markup language):

"violaBrief.hmml" Technical Overview of Viola (see the "docs" directory)

```
<!DOCTYPE hmml SYSTEM>
<HMML>
<TITLE>Viola, A Technical Summary</TITLE>
<CAUTION>THIS DOCUMENT IS IN DRAFT STATUS</CAUTION>
<HSEP>gold</HSEP>
<H1>VIOLA</H1>
<H3>Visual Interactive Object-oriented Langage and Applications</H3>
<P>
This paper presents a technical overview of viola.

<HSEP>gold</HSEP>
<H1>Overview</H1>
<P>
Viola is a tool for the development and support of interactive media
applications. Its basic functionality is not unlike that of
HyperCard and Tcl/Tk. Viola uses an object oriented model for
encapsulating data into ``object'' units, and to enforce a
classing and inheritance system. The extension language is
C-like in syntax, and is compiled into byte-code for
efficient interpretation. The graphical elements (widgets)
exist as classes in the Viola class hierarchy. The set of
widgets implemented in Viola are similar to those found in
graphical user interface toolkits like Xt, plus more
unusual widgets such as HyperCard-like cards and invisible
celopane buttons, and hypertext textfield.

<H2>Classes and Objects</H2>
<P>
The single inheritance classing system defines the basic types of
object instances. Many of these predefined class types happen to be
GUI oriented, because of the current application emphasis on hypermedia,
but many are non-visual and have nothing to do with GUIs. A modular object
model is enforced to control complexity: to provide a relatively simple
way of data encapsualization; for improving the size scalability of viola
applications; and for possibly helping network distribution of objects.
A scripting language exists for application writers to program
modifications to default object behaviors, and for application programming.
<P>
This is the Viola class hierarchy as of this writing.
It is rapidly evolving:
<VOBJF>../apps/chier.v</VOBJF>
```

<P>

This class hierarchy seems deficient (at this point and from some point of view, it's probably true) compared to the GUIs provided by toolkits like Motif. But, it's actually not as deficient as it seems. For the same reason as Tk, Viola does not require hard coding of, for example, dialog boxes to achieve the same functionality.

<P>

Because of the interpretive nature of the system, complex GUIs can be composed out of primitive elements, dynamically. To build a dialog box, a script could be written to create and necessary objects, and somehow combine them together to constitute a dialog box.

<P>

Making a dialog box can be made easy by calling a pre written procedure. The current way to do this in Viola is to build a ``dialog box maker object'', and to send to it a message:  
``Please make me a dialog box, with the following specifications''.

<H3>Hello, World!</H3>

<P>

Here's the proverbial <ITALIC>Hello World</ITALIC> program. Go ahead, click on it.

<VOBJF>../apps/violaBriefExample\_hello.v</VOBJF>

<P>And its file level representation:

<EXAMPLE>

```
\class {txtButton}
\name {violaBriefExample_hello}
\label {Hello, world!}
\script {
    switch (arg[0]) {
        case "buttonRelease":
            bell();
    }
    break;
}
usual();
}
\width {100}
\height {30}
\BGColor {grey45}
\BDColor {white}
\FGColor {white}
\
</EXAMPLE>
```

<P>

In reality the <CMD>switch()</CMD> would be busier than just handling one message. But it could just as easily be written thusly (and with non-essential color information left out):

<EXAMPLE>

```
\class {txtButton}
\name {hello}
\label {Hello, world!}
```

```
\script {  
    if (arg[0] == "buttonRelease") bell();  
    usual();  
}
```

```
\width {100}  
\height {30}  
</EXAMPLE>
```

<P>

Although it may seem that some simple binding mechanism would be less verbose, this free form allows one to easily compose the message handler in any order -- doing the default action first, then do the special thing, or any which way.

<H2>Messaging system</H2>

<P>

Viola is message driven, and messages may be generated by a number of sources. A message is typically caused by the user interacting with a graphical user interface object, but it could also be generated by other objects, or by a timer facility. Through a communication facility such as the socket, a message may also be generated from another process on the network.

<P>

In the above ``Hello, world!'' example, when the button is clicked on, that button object "hello" will eventually receive a "buttonRelease" message, which according to the script will execute the <CMD>bell()</CMD> command. If the object does not have any message handlers, the message will ``fall thru'' the object, and (by way of <CMD>usual()</CMD>) the class default action will occur.

<P>

A typical viola application consists of a collection of objects interacting -- generating, receiving, and delegating messages -- with each other, and with the user.

<H2>The Extension Language</H2>

<P>

As seen in the example above, viola scripts are C-like in syntax. The language supports way few constructs: <CMD>if, while, for, switch</CMD>. The commands like <CMD>print(), exit(), create()</CMD>, etc are all implemented as <ITALIC>methods</ITALIC>. Instead of building the commonly used commands into the language grammar, they actually are just defined early enough in the class hierarchy as to be accessible by all subclasses that may need them.

<P>

All objects can be individually programmed using the scripting language. Each object is essentially its own interpretive environment, and each object is its own variable scope.

<P>

For optimization, object scripts are compiled into <ITALIC>byte codes</ITALIC> before applying the byte code interpreter on them. Because an object's script is basically a message event handler that is likely to receives many messages in its instance life time, the one time

cost of parsing and simple transformation into byte codes is very worthwhile. The gain in execution speed is especially apparent when the objects deal with time critical "mouseMove" messages, or if there are tight looping operations.

<HSEP>gold</HSEP>  
<H1>Applications</H1>  
<P>

Along side the development of the Viola language/toolkit <ITALIC>engine</ITALIC> itself, there is also the development of real working applications using the engine. The two processes provide reality checks for each other.

<P>  
Here we show screendumps of two developing viola applications.

<H2>World Wide Web Browser</H2>  
<FIGURE TYPE="image/gif" SRC="../../../docs/violaWWW.gif">  
<P>

This ``ViolaWWW'' application is currently among the most actively developed viola application. The initial viola-WWW effort was made in order to provide to viola a clean network transport mechanism. But the ViolaWWW browser application itself turned out to be useful enough, that it is being actively developed, with emphasis on support for online publishing.

<P>  
An early version of this browser has been in use in the WWW community since mid 92, it being the first publically available World Wide Web browser for X-Windows.

<H2>The Whole Internet Resource Catalog</H2>  
<FIGURE TYPE="image/gif" SRC="twi.gif">  
<P>

An electronic version of the resource catalog portion of the book <ITALIC>The Whole Internet</ITALIC>. This application uses HyperCard style card-flipping technique to flip among four basic GUI sets (the cover frame is shown here; others frames contain documents and controlling GUI elements).

<HSEP>gold</HSEP>  
<H1>Summary</H1>  
<P>

In sum, the Viola language/toolkit system provides an environment where applications are composed of groups of objects, where objects interact, by message passing, with the user and with each other.

<P>  
As more applications are developed, more reusable objects will be created. And development of successive applications will become easier and easier. One of the goals of the Viola project is to accumulate a collection of objects useful for constructing hypermedia applications.

<P>  
The immediate future direction of Viola development will continue to aim towards the path of hypermedia applications, with the World Wide Web as the document/object network transport infrastructure.



```
<P>
If you're interested in contributing to the development effort,
please contact me.
<HSEP>gold</HSEP>
<ADDRESS>
<P>Pei Y. Wei
<P>Developer, O'Reilly & Associates, Digital Media Group
<P><CMD>wei@ora.com</CMD>
</ADDRESS>
</HMML>
```

The contents of file "violaCh1.hmml" (contained in the "docs" directory):

```
<!DOCTYPE hmml SYSTEM
[
]>
<HMML>
<SECTION NAME="chapter1">
<H1>Introduction to Viola</H1>
<FIGURE TYPE="image/xbm" SRC="viola.xbm">

<SECTION NAME="whatIsViola">
<H2>What is Viola?</H2>
<P>
Viola is a hypermedia application authoring and supporting system.
It contains a graphical user interface set, an ``object oriented''
data organization and storage model, and a built-in extension
language. Perhaps the most important contribution of Viola is its
potential in bringing HyperCard-like capability to a very wide
range of platforms.
<p>
Viola can be used for the development and support of
interactive media applications. It provides an object data
organization model, an interpreted extension language,
graphical elements for user interface. The Viola operating
environment is interpretive, designed for rapid prototyping.
<P>
Viola is desigend to aid the development and support of
interactive/hyper media applications for the Unix/X platform.
Its functionality is similar to HyperCard and Tcl/Tk.
Viola uses an object oriented model to facilitate data
encapsulation into ``object'' units, and to enforce a
classing and inheritance system. The extension language is
C-like in syntax and is processed into byte-code for
efficient interpretation. The graphical elements (widgets)
exist as classes in the Viola class hierarchy. The set of
widgets implemented in Viola are similar to those found in
user interface toolkits like Xt, plus more unusual widgets
such as HyperCard-like cards and invisible celopane buttons,
and hypertext textfield.
```

<P>

In sum, Viola provides an environment in which applications are composed of groups of objects where each object interacts, by message passing, with the user and with each other.

<P>

Because most aspects of an object is accessible and controllable through the interpreted extension language, building an application in Viola can be done dynamically, without the edit/compile cycle. As with other systems with built-in extension language (Emacs/ELisp, Tk/Tcl, HyperCard/HyperTalk), Viola derives much of its versatility from its extension language.

<P>

The rest of this paper gives a brief overview of the Viola basics: the object model, language, and GUI elements. It also describes some applications(?).

</SECTION>

<SECTION NAME="objectSystem">

<H2>The Object System</H2>

<P>

This section briefly describes Viola's notion of object orientation.

<P>

Each Viola object consists of an array of ``slot'' values. These values are information pertaining specifically to the object: its class, name, script, color, and so on. The number and type of each slot in an object are determined by the class of the object.

<P>

Each class inherits slot definitions from its superclasses, and has the option to set new values for the inherited slots. In addition to those inherited slots, it may define two types of new slots: private and common.

<P>

Common slots define slots that are shared by all object instances of the same class. Private slots define slots that make up each object instance. The separation of common and private slots reduces redundancy of information carried by each object.

<P>

As with slots, class methods are also inherited. The idea, again, is to provide a mechanism for sharing as much code as possible. It also makes the task of subclassing relatively easy and systematic. It should be noted that modification of the object system (to subclass, adding slots and methods) must, at this point, be done in C.

<P>

This is the Viola class hierarchy as of this writing. It is evolving rapidly.

</SECTION>

<SECTION NAME="violaClassHierarchy">  
<H2>The Viola Class Hierarchy</H2>  
<EXAMPLE>

```
    cosmic
      generic
        field
          BCard
          FCard
          XBM
          XBMBUTTON
          toggle
          XPM
          XPMBUTTON
          GIF
          dial
          client
          TTY
          socket
          menu
          pane
          hpane
            txt
            txtLabel
            txtButton
            txtDisp
            txtEdit
          vpane
          project
          rubber
          slider
          stack
          tray
```

</EXAMPLE>

<P>

The cosmic class defines the minimal object: a private slot that lets the object know what class it belongs to; and essential methods such as create(), destroy(), save(), etc. From here on the slots and methods definition is rather arbitrary and depends on what the application is.

<P>

As Viola was designed for visually interactive applications, most of the classes are GUI widgety oriented. The two notable exceptions are the socket and TTY classes, which are useful for communicating with other processes.

<P>

The class hierarchy seems deficient (at this point and from some point of view, it's probably true) compared to the GUIs provided by toolkits like Motif. But, it's actually not as deficient as it seems. For the same reason as Tk, Viola does not require hard coding of, for example, dialog boxes to achieve the same functionality.

<P>

Because of the interpretive nature of the system, complex GUIs can be composed out of primitive elements, dynamically. To build a dialog box, a script could be written to create and necessary objects, and somehow combine them together to constitute a dialog box.

<P>

As in Tk, making a dialog box can be made easy by calling a pre written procedure. The current way to do this in Viola is to build a ``dialog box maker object'', and to send to it a ``Please make me a dialog box, with the following specifications''.

<P>

It's worthwhile to illustrate with an example, which will show many other aspects of Viola.

</SECTION>

<SECTION NAME="hello.v">

<H2>hello.v</H2>

<EXAMPLE>

```
\class {txtButton}
\name {hello}
\label {Hello, world!}
\script {
  switch (arg[0]) {
    case "buttonRelease":
      res.dialog("show",
                "Are you sure you want to exit?",
                "Yes",      "callback_exit",
                "No",       "callback_nevermind");
      break;
    case "callback_exit":
      exit(0);
    case "callback_nevermind":
      return; /* do nothing */
  }
  usual();
}
```

</EXAMPLE>

</SECTION>

</SECTION>

</HMML>

Reexamination/Control Number:  
90/006,831  
Art Unit: 3992

Page 73

**How to Contact the Examiner:**

Any inquiry concerning this communication or earlier communications from the examiner should be directed to St. John Courtenay III, whose telephone number is 571-272-3761. A voice mail service is also available at this number. The Examiner can normally be reached on Monday - Friday, 9:00 AM - 5:00 PM.

If attempts to reach the examiner by telephone are unsuccessful, the Examiner's Supervisor, Mark Reinhart who can be reached at 571-272-1611.

The fax phone number for the organization where this application or proceeding is assigned is:


**CENTRAL REEXAMINATION UNIT FAX NUMBER:**

**571-273-9900**

**All responses sent by U.S. Mail should be mailed to:**

Commissioner for Patents  
PO Box 1450  
Mail Stop ex parte REEXAM  
Alexandria, VA 22313-1450

*Conference:  
w/ Reinhart SPRE CRU 3992*

  
ST. JOHN COURTENAY III  
PRIMARY EXAMINER

EOLASTX-0000001372



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE  
 United States Patent and Trademark Office  
 Address: COMMISSIONER FOR PATENTS  
 P.O. Box 1450  
 Alexandria, Virginia 22313-1450  
 www.uspto.gov



Bib Data Sheet

CONFIRMATION NO. 9718

SERIAL NUMBER 90/006,831	FILING DATE 10/30/2003  RULE	CLASS 709	GROUP ART UNIT 3992	ATTORNEY DOCKET NO.
-----------------------------	---------------------------------------	--------------	------------------------	---------------------

APPLICANTS

5838906, Residence Not Provided;

University of California(Owner), Alameda, CA;  
 Director Ordered Reexam, Alexandria, VA;

\*\* CONTINUING DATA \*\*\*\*\*

This application is a REX of 08/324,443 10/17/1994 PAT 5,838,906

*S.J.L*

\*\* FOREIGN APPLICATIONS \*\*\*\*\*

*ST.JL*

Foreign Priority claimed 35 USC 119 (a-d) conditions met Verified and Acknowledged	<input type="checkbox"/> yes <input checked="" type="checkbox"/> no <input type="checkbox"/> yes <input checked="" type="checkbox"/> no <input type="checkbox"/> Met after Allowance Examiner's Signature <i>[Signature]</i> Initials <i>[Initials]</i>	STATE OR COUNTRY	SHEETS DRAWING	TOTAL CLAIMS 10	INDEPENDENT CLAIMS 3
--	---	---------------------	-------------------	-----------------------	----------------------------

ADDRESS

30080  
 LAW OFFICE OF CHARLES E. KRUEGER  
 P.O. BOX 5607  
 WALNUT CREEK , CA  
 94596-1607

TITLE

DISTRIBUTED HYPERMEDIA METHOD FOR AUTOMATICALLY INVOKING EXTERNAL APPLICATION PROVIDING INTERACTION AND DISPLAY OF EMBEDDED OBJECTS WITHIN A HYPERMEDIA DOCUMENT

FILING FEE RECEIVED 0.00	FEES: Authority has been given in Paper No. _____ to charge/credit DEPOSIT ACCOUNT No. _____ for following:	<input type="checkbox"/> All Fees <input type="checkbox"/> 1.16 Fees ( Filing ) <input type="checkbox"/> 1.17 Fees ( Processing Ext. of time ) <input type="checkbox"/> 1.18 Fees ( Issue ) <input type="checkbox"/> Other _____ <input type="checkbox"/> Credit
-----------------------------	---	---



# 90/006,831 REEXAM SEARCH WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Thursday, August 25, 2005 *ST.JC*

*John Courtenay III*  
8-25-2005  
ST. JOHN COURTENAY III  
PRIMARY EXAMINER

Hide?	Set Name	Query	Hit Count
	<i>DB=PGPB,USPT; PLUR=YES; OP=OR</i>		
<input type="checkbox"/>	L35	l28 or L34	1
<input type="checkbox"/>	L34	l32 and L33	1
<input type="checkbox"/>	L33	(client\$1 and server\$1 and (embed\$3 adj text) and hypermedia and browser\$1 and l25 and l26 and l27).clm	1
	<i>DB=USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR</i>		
<input type="checkbox"/>	L32	l28 and L31	1
<input type="checkbox"/>	L31	l29 or l30	9897
<input type="checkbox"/>	L30	l11 or l12 or l13 or l14 or l15 or l16 or l17 or l18 or l19 or l20	3309
<input type="checkbox"/>	L29	l1 or l2 or l3 or l4 or l5 or l6 or l7 or l8 or l9 or l10	6689
<input type="checkbox"/>	L28	l21 and l22 and l24 and l25 and l26 and L27	2
<input type="checkbox"/>	L27	interactive and browser\$1 and l23	433
<input type="checkbox"/>	L26	object\$1 or (data adj object\$1) same embedded	4656119
<input type="checkbox"/>	L25	executable adj (application\$1 or program\$1 or code\$1)	11456
<input type="checkbox"/>	L24	browser\$1	30663
<input type="checkbox"/>	L23	hypermedia	1614
<input type="checkbox"/>	L22	embed\$3 adj text	350
<input type="checkbox"/>	L21	client\$1 and server\$1	73723
<input type="checkbox"/>	L20	345/656.ccls.	67
<input type="checkbox"/>	L19	345/655.ccls.	12
<input type="checkbox"/>	L18	345/654.ccls.	17
<input type="checkbox"/>	L17	345/653.ccls.	38
<input type="checkbox"/>	L16	345/649.ccls.	147
<input type="checkbox"/>	L15	345/638.ccls.	20
<input type="checkbox"/>	L14	345/619.ccls.	664
<input type="checkbox"/>	L13	345/427.ccls.	592
<input type="checkbox"/>	L12	345/419.ccls.	1640
<input type="checkbox"/>	L11	719/310.ccls.	423
<input type="checkbox"/>	L10	715/777.ccls.	72
<input type="checkbox"/>	L9	715/760.ccls.	170
<input type="checkbox"/>	L8	715/526.ccls.	344



# L28 Hit List

Search Results - Record(s) 1 through 2 of 2 returned.

1. Document ID: US 5838906 A

Using default format because multiple data bases are involved.

L28: Entry 1 of 2

File: USPT

Nov 17, 1998

US-PAT-NO: 5838906

DOCUMENT-IDENTIFIER: US 5838906 A

TITLE: Distributed hypermedia method for automatically invoking external application providing interaction and display of embedded objects within a hypermedia document

DATE-ISSUED: November 17, 1998

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Doyle; Michael D.	Alameda	CA		
Martin; David C.	San Jose	CA		
Ang; Cheong S.	Pacifica	CA		

US-CL-CURRENT: 709/202; 709/218, 715/513, 715/515, 715/516, 715/738, 715/804, 719/315

Full	Title	Citation	Front	Review	Classification	Date	Reference	Claims	KBWC	Draw Ds
------	-------	----------	-------	--------	----------------	------	-----------	--------	------	---------

2. Document ID: US 5838906 A

L28: Entry 2 of 2

File: DWPT

Nov 17, 1998

DERWENT-ACC-NO: 1999-023910

DERWENT-WEEK: 200377

COPYRIGHT 2005 DERWENT INFORMATION LTD

TITLE: Interactive hypermedia access method for Internet - involves executing external application associated with hypermedia document on client workstation

Full	Title	Citation	Front	Review	Classification	Date	Reference	Claims	KBWC	Draw Ds
------	-------	----------	-------	--------	----------------	------	-----------	--------	------	---------

# L32-L35 Hit List

Search Results - Record(s) 1 through 1 of 1 returned.

1. Document ID: US 5838906 A

Using default format because multiple data bases are involved.

L32: Entry 1 of 1

File: USPT

Nov 17, 1998

US-PAT-NO: 5838906

DOCUMENT-IDENTIFIER: US 5838906 A

TITLE: Distributed hypermedia method for automatically invoking external application providing interaction and display of embedded objects within a hypermedia document

DATE-ISSUED: November 17, 1998

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Doyle; Michael D.	Alameda	CA		
Martin; David C.	San Jose	CA		
Ang; Cheong S.	Pacifica	CA		

US-CL-CURRENT: 709/202; 709/218, 715/513, 715/515, 715/516, 715/738, 715/804, 719/315

Full	Title	Citation	Front	Review	Classification	Date	Reference			Claims	KBAC	Draw D
------	-------	----------	-------	--------	----------------	------	-----------	--	--	--------	------	--------

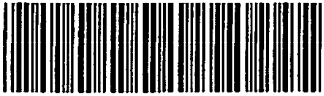
Terms	Documents
L28 and L31	1

Display Format:

[Previous Page](#)   [Next Page](#)   [Go to Doc#](#)

<input type="checkbox"/>	L7	715/516.ccls.	110
<input type="checkbox"/>	L6	715/513.ccls.	1050
<input type="checkbox"/>	L5	718/106.ccls.	368
<input type="checkbox"/>	L4	709/203.ccls.	2814
<input type="checkbox"/>	L3	709/219.ccls.	1930
<input type="checkbox"/>	L2	709/200.ccls.	711
<input type="checkbox"/>	L1	715/501.1.ccls.	680

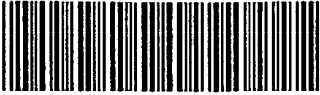
END OF SEARCH HISTORY

<b>Reexamination</b> 	<b>Application/Control No.</b> 90/006,831	<b>Applicant(s)/Patent Under Reexamination</b> 5838906
	<b>Certificate Date</b>	<b>Certificate Number</b>

<b>Requester</b> <b>Correspondence Address:</b> <input type="checkbox"/> <b>Patent Owner</b> <input type="checkbox"/> <b>Third Party</b>
Director of Patents P.O. Box 1450 Alexandria VA 22313-1450

<b>LITIGATION REVIEW</b> <input checked="" type="checkbox"/>	<i>ST,JC</i> <small>(examiner initials)</small>	<b>8-24-05</b> <small>(date)</small>
<b>Case Name</b>		<b>Director Initials</b>
(1) Eolas Technologies Inc. and The Regents of The University of California v. Microsoft Corp.,		
U.S. District Court, N. District of Illinois, Eastern Division, Docket No. 99 C 0262, decided Jan. 14, 2004.		
(2) Eolas Technologies Inc. and The Regents of The University of California v. Microsoft Corp.,		
U.S. Court of Appeals for the Federal Circuit, Docket No. 04-1234, decided March 2, 2005.		

<b>COPENDING OFFICE PROCEEDINGS</b>	
<b>TYPE OF PROCEEDING</b>	<b>NUMBER</b>
1. None	
2.	
3.	
4.	

<b>Issue Classification</b> 	Application/Control No.	Applicant(s)/Patent under Reexamination	
	90/006,831	5838906	
	Examiner	Art Unit	
	St. John Courtenay III	3992	

ISSUE CLASSIFICATION												
ORIGINAL					CROSS REFERENCE(S)							
CLASS		SUBCLASS			CLASS	SUBCLASS (ONE SUBCLASS PER BLOCK)						
715		501.1			709	203	219					
INTERNATIONAL CLASSIFICATION					718	106						
G	O	6	F	9/54	715	513	516	526	760	777		
				/	719	310						
				/	345	419	427	619	638	649	653 654	
				/	345	655	656					
				/								

_____ (Assistant Examiner) (Date)		<b>ST. JOHN COURTENAY III</b> <b>PRIMARY EXAMINER</b>  St. John Courtenay III <i>St. John Courtenay III</i> (Primary Examiner) (Date) 8-24-05		Total Claims Allowed: 10	
_____ (Legal Instruments Examiner) (Date)				O.G. Print Claim(s) 1	O.G. Print Fig. 9

<input checked="" type="checkbox"/> Claims renumbered in the same order as presented by applicant												<input type="checkbox"/> CPA		<input type="checkbox"/> T.D.		<input type="checkbox"/> R.1.47	
Final	Original	Final	Original	Final	Original	Final	Original	Final	Original	Final	Original	Final	Original	Final	Original		
	1		31		61		91		121		151		181				
	2		32		62		92		122		152		182				
	3		33		63		93		123		153		183				
	4		34		64		94		124		154		184				
	5		35		65		95		125		155		185				
	6		36		66		96		126		156		186				
	7		37		67		97		127		157		187				
	8		38		68		98		128		158		188				
	9		39		69		99		129		159		189				
	10		40		70		100		130		160		190				
	11		41		71		101		131		161		191				
	12		42		72		102		132		162		192				
	13		43		73		103		133		163		193				
	14		44		74		104		134		164		194				
	15		45		75		105		135		165		195				
	16		46		76		106		136		166		196				
	17		47		77		107		137		167		197				
	18		48		78		108		138		168		198				
	19		49		79		109		139		169		199				
	20		50		80		110		140		170		200				
	21		51		81		111		141		171		201				
	22		52		82		112		142		172		202				
	23		53		83		113		143		173		203				
	24		54		84		114		144		174		204				
	25		55		85		115		145		175		205				
	26		56		86		116		146		176		206				
	27		57		87		117		147		177		207				
	28		58		88		118		148		178		208				
	29		59		89		119		149		179		209				
	30		60		90		120		150		180		210				

<b>Ex Parte Reexamination Interview Summary</b>	<b>Control No.</b> 90/006,831	<b>Patent Under Reexamination</b> 5838906	
	<b>Examiner</b> St. John Courtenay III	<b>Art Unit</b> 3992	

All participants (USPTO personnel, patent owner, patent owner's representative):

- (1) St. John Courtenay III (3) Michael D. Doyle  
(2) Mark Reinhart (4) Charles Krueger

Date of Interview: 18 August 2005

Type: a)  Telephonic b)  Video Conference  
c)  Personal (copy given to: 1)  patent owner 2)  patent owner's representative)

Exhibit shown or demonstration conducted: d)  Yes e)  No.  
If Yes, brief description: Powerpoint presentation of Patent Owner's arguments.

Agreement with respect to the claims f)  was reached. g)  was not reached. h)  N/A.  
Any other agreement(s) are set forth below under "Description of the general nature of what was agreed to..."

Claim(s) discussed: 1 and 6.

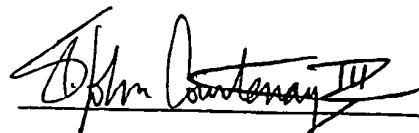
Identification of prior art discussed: Mosaic (APA), Berners-Lee, Raggett I & II, and Tove.

Description of the general nature of what was agreed to if an agreement was reached, or any other comments:  
The Patent Owner presented a Powerpoint presentation summarizing the Patent Owner's arguments of record. The Examiner informed the patent owner that OPLA was reviewing the Viola code to determine if it should be considered as a prior art publication.

(A fuller description, if necessary, and a copy of the amendments which the examiner agreed would render the claims patentable, if available, must be attached. Also, where no copy of the amendments that would render the claims patentable is available, a summary thereof must be attached.)

**A FORMAL WRITTEN RESPONSE TO THE LAST OFFICE ACTION MUST INCLUDE PATENT OWNER'S STATEMENT OF THE SUBSTANCE OF THE INTERVIEW. (See MPEP § 2281). IF A RESPONSE TO THE LAST OFFICE ACTION HAS ALREADY BEEN FILED, THEN PATENT OWNER IS GIVEN ONE MONTH FROM THIS INTERVIEW DATE TO PROVIDE THE MANDATORY STATEMENT OF THE SUBSTANCE OF THE INTERVIEW (37 CFR 1.560(b)). THE REQUIREMENT FOR PATENT OWNER'S STATEMENT CAN NOT BE WAIVED. EXTENSIONS OF TIME ARE GOVERNED BY 37 CFR 1.550(c).**

ST. JOHN COURTENAY III  
PRIMARY EXAMINER

  
Examiner's signature, if required

cc: Requester (if third party requester)