Beyond the Browser

Rediscovering the Role of the Desktop in a Net-centric World

Richard Gaskin
Fourth World Systems

First draft published: 20 September, 2001
Last revised: 17 April, 2009

French Translation available in PDF,
generously progided by René Micout

Abstract
While the Web browser may be ideally suited for viewing data, its design is not optimized for creating or manipulating data. With the advent of Web applications, many interface designers have been limiting their work to those systems which can be delivered in a Browser window. Using a wider range of tools, technologies, and protocols, a designer may find some tasks better served through Net-aware desktop applications.

Keywords: web applications, application design, distributed computing, Internet protocols, scripting, systems programming, user interface design.

"Many articles have been published extolling the virtues of Browser-based applications, but few (if any) of these were written in one..."

1. Introduction: The Internet is not the Web

The Internet is used for a wide variety of tasks, from transferring files to sending email. These tasks also include viewing text, graphics, and rich-media content using the World Wide Web. While the Web is a relatively recent newcomer to the range of Internet services, its popularity has eclipsed the other services of the Internet. Today, many lay people think "the Web" and "the Internet" are synonymous.

With the growing ubiquity of the Web, many application designers have felt that the next logical step is to

migrate tasks from desktop applications to the Web Browser window. Some tasks are well suited for the Browser, taking advantage of users' confidence in the inherent simplicity of the Browser environment.

But this same simplicity also introduces unique challenges, from interface layout to handling protocols other than HTTP, which can be handled well in dedicated applications.

Word processing and many other traditional computing applications involve tasks which are essentially solitary in nature; the files may benefit from being shared among, and even modified by, multiple users, but the nature of the task itself (crafting a spreadsheet, retouching a photo, writing a letter) is generally something one does by themselves.

Then there are tasks which are inherently group-oriented, which take on additional value when connected to multiple computers. Email, for example, has little practical value without other users involved. It is for these connected-computing tasks that the Internet is especially useful, and only a subset of these tasks is optimal when implemented inside a Browser window.

The term "Web app" has been used to describe applications that live inside a Browser window. This paper introduces the term "Net app" to describe Internet-connected applications implemented as independent processes with their own user interface.

Figure 1 shows a conceptual illustration of various application types. In a modern GUI, all applications are desktop applications, as the desktop is the primary object serving as an enclosure for the user experience. A subset of applications are network applications (Net Apps), which use Internet protocols to provide the benefits unique to distributed computing. The Web Browser is one of these, but there are many others, including email, FTP clients, instant messaging, etc.

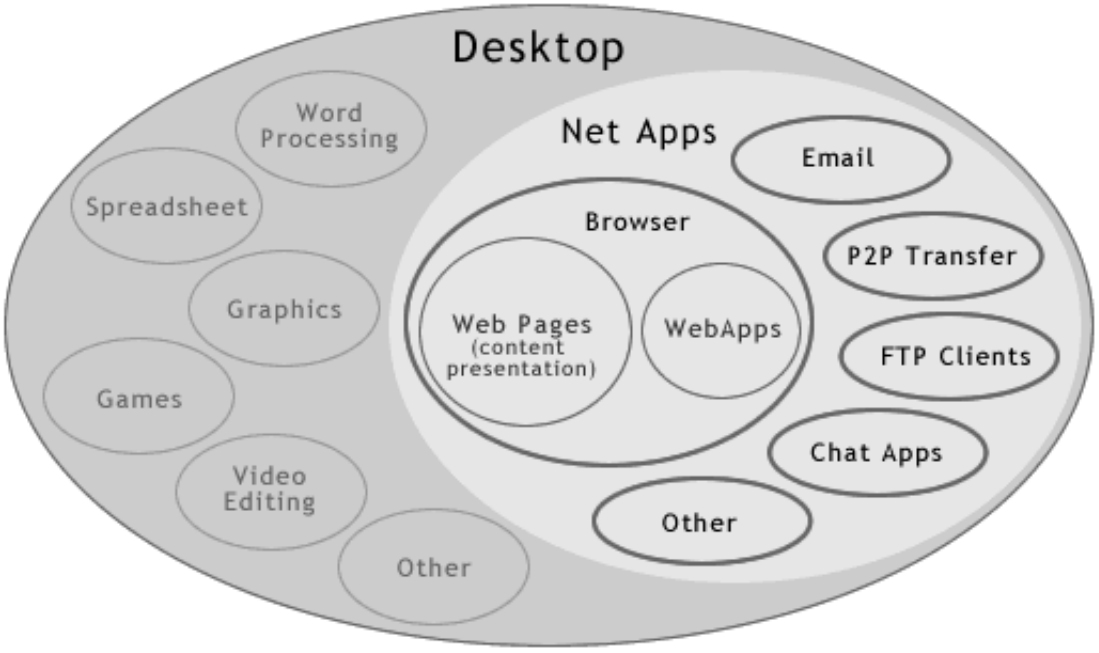

Figure 1: Categories of software applications, emphasizing subcategories of network apps.

While the Browser will remain a useful option for network applications, the growing popularity of peer-to-peer (P2P) tools like Naptster and the Gnutella network, and messaging tools like AOL Instant Messenger and ICQ, suggest that dedicated applications may be more useful for some tasks.

2. Inside the Browser or Outside: Viewing vs. Authoring

When deciding among alternatives for implementing connected-computing applications, it is useful to

consider the nature of the task the application will support. The Browser presents unique strengths and weaknesses, and not all tasks are optimally supported once we examine its core functionality.

Some guidance can be found in the origins of the Browser, examining the specific range of tasks it was designed to support. The Web was invented as a means of allowing researchers to view documents created by other researchers. With the invention of the graphical Browser these documents could also include graphics and other media. The controls provided in the Browser interface reflect this passive-viewing orientation, focused as they are on tasks involving navigation among Web pages.

Later JavaScript was added to Netscape Navigator and JScript to Microsoft Internet Explorer, making it possible to add interactivity to Web pages. The scripting languages allow both client-side applications like calculators as well as client-server applications such as submitting forms to a CGI application.

The range of user interactions within the Browser has expanded even more with tools like Java and Macromedia Flash. While interfaces created with these technologies may be placed within a Web page they are not fully integrated into the Browser experience. and the disparity between how Browsers behave with HTML-only pages and Flash- or Java-enhanced pages may cause confusion for the user. For example, using the Browser's Back button while viewing Flash media will not take the user to the previous screen layout in Flash, but in most cases it will instead completely remove the Flash element altogether as it loads the previously-viewed page in its place. For this reason, many developers using Java and Flash do so in secondary popup windows which hide the navigation controls, effectively attempting to emulate a desktop application experience within the confines of the Browser.

Such attempts to provide the interaction functionality common in desktop applications meet with a number of limitations. In addition to the disparity between the Browser's inherent navigation functionality and that of the applet residing within it, the designer also faces these considerations:

Controls - Layout

One example of limitations in designing applications in the Browser is dynamic composition, designing control layouts that adjust themselves to fit in windows which can be resized by the user. While common in desktop applications, dynamic compositions are difficult to implement in a Browser. AOL's New Email form is a good case in point, shown in Figure 2.

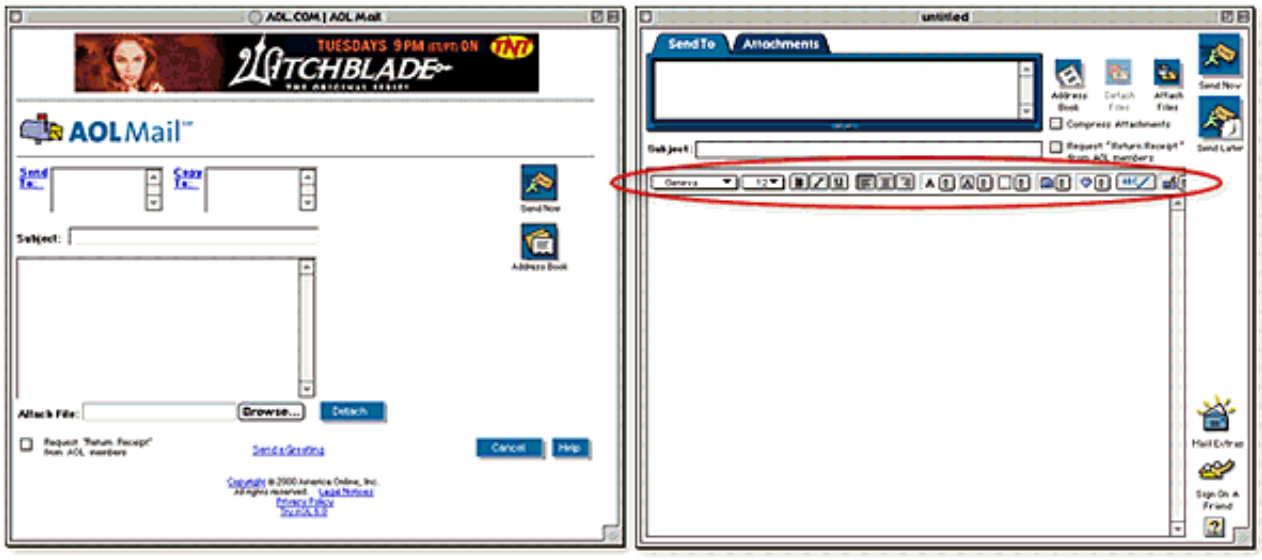

Figure 2: AOL New Email form, shown in the Web version (left) and in the Desktop software.

The limitations of HTML do not allow the primary content region (where the user types their email message) to resize to match the size the user has selected for the window.

Also note that significantly less screen space is devoted to the task the interface ostensibly supports. More than half of the non-task space is altogether unused, with the rest comprised largely of advertisements and other links designed to take the user away from the task at hand. While common in the Web experience, such elements serve to distract from the goal of the page, effectively becoming “anti-features”.

Controls - Behavior

The two implementations of AOL's email interface illustrate another limitation in designing applications in the Browser: many features and controls are difficult or impossible to create using HTML and JavaScript.

While AOL's Web team did a good job of approximating the control layout and essential functionality of the Desktop application, note that the Desktop version supports an array of text styling attributes not present in the Browser-based version (Figure 2, outlined in red).

Other Desktop features not supported in the Web interface include the ability to compress attachments, to have more than one email attachment, and to save an email message without sending it.

Another small but pervasive example of unusual behavior of controls in the Browser is the target area of radio and checkbox controls on Web pages. In desktop applications, both the label and indicator portions of the control are active areas which respond to the user's mouse click, but most Browser implementations limit the target area to the indicator only, as illustrated in Figure 3. This measurably hampers its usefulness according to Fitt's Law, which suggests that "the time to acquire a target is a function of the distance to and size of the target". Only some of the most recent Browsers allow the option for standard target areas for radio controls and checkboxes, and merely upgrading the Browser does not fix the problem; pages must be rewritten to employ the correct behavior.

Figure 3: Radio controls, showing the target area outlined in red.

Menus

And while users are accustomed to finding commands in the menu bar, in a Browser the menu bar belongs exclusively to the Browser itself. Without the ability for Web apps to modify the Browser's menu bar, designers have adopted a wide range of workaround pull-down menus written in JavaScript. These workaround menus employ a dismayingly broad range of interaction behaviors with little or no consistency between applications, and almost never behave like the "normal" menu bar users are already familiar with. These inconsistencies are compounded by the broadly different implementations of the JavaScript language, as any UNIX or Mac OS user can tell you about their experiences at sites developed exclusively for one platform.

File I/O

The ability to read and write data to files is essential for any application that requires persistent information across sessions. Attempting a modest level of security, Browsers do not normally allow Web pages, or even plug-ins, the ability to read and write to disk. Instead the Browser provides *cookies*, a mechanism for storing only a small about of data on the client machine which can be used to lookup significant data stored on the server. Cookies requires

multiple HTTP transactions for each discrete data element delivered to the interface, while a desktop application can access even large amounts of data in milliseconds from local storage.

Responsiveness

In addition to the latency introduced by the lack of file I/O support, Web apps are made even less responsive than their desktop counterparts by requiring the interface to be downloaded every time it is used. In contrast, the AOL client provides an excellent balance between downloaded and local components, storing consistent user interface elements on the client machine and downloading new interface elements and timely content as needed.

Protocols

While the Internet supports a wide variety of protocols for different tasks, the Browser primarily handles only HTTP natively, with some limited support for FTP. Other protocols like SMTP (email), NNTP (newsgroups), Gnutella (file sharing), IRC (chat), and others require Java or a plug-in to implement inside the Browser, and are more commonly supported in dedicated applications.

Given these considerations, we can see that some applications can work well inside a Browser window, but other tasks can be clumsy or even altogether impractical when attempted outside of an environment dedicated to that task. Many articles have been published extolling the virtues of Browser-based applications, but few (if any) of these were written in one; most were written in a dedicated word processor.

Tasks like word processing, graphics production, and other media-creation tasks provide a useful distinction for the domain of dedicated applications: they involve *creating* data. In contrast, successful Browser applications are more commonly oriented toward the *viewing* of data.

3. Security Considerations

One of the perceived benefits of choosing to implement an application in a Browser is security. By restricting file I/O, and through encrypted transmissions via protocols like HTTPS, browsers offer a level of security that is perceived as generally acceptable to even large corporations.

However, few security restrictions exist for embedded objects like ActiveX controls and Director XTRAs, and such objects are not unusual on the Web.

In most Browsers, by default the user must explicitly confirm the download and installation of such objects. But doing so changes the discussion of security from anything unique to the browser, and reframes it as a matter of the trust the user feels for the site providing the embedded object.

Given the growing prevalence of executable code transferred over the Internet, and as illustrated by such OS-specific viruses as Code Red and SirCam, the Browser's role in ensuring security is apparently limited. While many corporations choose Web applications for the perceived security, most of the corporations also choose the Windows family of operating systems, where increasing exploitation of security holes unique to that OS family cost those corporations US$3.6 billion cleaning up after Code Red and SirCam alone.

In terms of actual security, the informed user should have as much confidence downloading an application from a trusted provider as they would any other object code such as an ActiveX control. Net apps can use the same protocols as Browser applications, including HTTPS, to provide similar levels of security for outgoing data.

In practical terms, the risks of running a standalone Net application are not significantly greater than for many embedded objects commonly included in Web pages.

4. Options for Net App Capabilities: More than P2P

Once we begin to look for development opportunities outside the Browser, we find that most of the press devoted to such alternatives focuses on peer-to-peer file sharing (P2P) exclusively. While Napster and GNUtella have indeed earned significant attention, there's so much more to the Internet than file sharing.

One of the biggest growth categories over the last year has been instant messaging, with tools like AOL Instant Messenger (AIM), Microsoft Messenger, Yahoo Messenger, Hotline, ICQ, and others. In addition to providing real-time text exchange between users, most of these applications also provide group chat, file sharing, and other services.

And "push technology", in which users can choose to have specific information sent to them instead of going on to the Web to get it, seems to be on a comeback. While initial proponents of push like Marimba focused on marketing uses, push technology can be more valuable as a business tool, delivering timely rich-media content to employees.

When designing custom Net apps, you can mix and match these and other technologies in much more flexible ways than can be achieved in the Browser. Table 1 provides a brief listing of popular Internet protocols, noting those supported in the most popular Browsers:

Protocol	Description	Browser-Supported
HTTP	HyperText Transfer Protocol (Web)	Yes
FTP	File Transfer Protocol (File transfer)	Limited
SMTP	Simple Mail Transfer Protocol (Email)	No
NNTP	Network News Transport Protocol (Newsgroups)	No
IRC	Internet Relay Chat (Chat)	No
GNUtella	(File transfer, distributed computing)	No
Custom	New protocols can be developed as needed	No

Table 1: Popular Internet protocols

5. Tools for Creating Net Apps

One of the key factors which have contributed to the success of the Web is its inherent platform-independent nature: users can have a similar experience regardless of which operating system they use. Accordingly, any tools used for building Net apps should also run on as many platforms as possible.

Even corporate intranets, where internal staff often determines specific machine configurations, can benefit from platform independence. It's not uncommon for large corporations to have mostly Windows-based systems, but also have UNIX servers and Macintosh clients in some departments.

Given the many significant differences between OS families, developing multi-platform applications in traditional languages like C++ is cost-prohibitive for all but the largest organizations. In response, new platform-independent languages like Java have emerged.

But for the reasons John K. Osterhaut describes in his paper, Scripting: Higher Level Programming for the 21st Century, there are many benefits to using fourth-generation languages (4GLs) for developing Net apps. A number of cross-platform tools exist to allow extremely rapid development of Net apps.

Table 2 offers a brief listing of available development tools that provide access to Internet protocols and provide deployment on at least two platforms:

Development Tool with links to vendor	Distribution Platforms		
	UNIX/Linux	Win32	Mac OS
Java Sun Microsystems	Yes	Yes	Yes
Adobe AIR Adobe Systems	Yes	Yes	Yes
Tcl/Tk Scriptics	Yes	Yes	Yes
REALBasic REAL Software, Inc.		Yes	Yes
MetaCard MetaCard Corporation	Yes	Yes	Yes
Revolution Runtime Revolution Ltd.	Yes	Yes	Yes
Macromedia Flash MX Macromedia, Inc.	Yes (Player only)	Yes	Yes
Macromedia Director Macromedia, Inc.		Yes	Yes
iShell Tribeworks		Yes	Yes
Rebol REBOL Technologies	Yes	Yes	Yes
Ruby Yukihiro Matsumoto	Yes	Yes	

Table 2: Net App development tools and languages

If you have a favorite Net app development tool or language not listed here, please let me know.

6. Resources

Below is a collection of additional sources of information related to Net apps. You can suggest additions to this list by emailing the URL to netapps2007@fourthworld.com.

Articles

Good Riddance to Browser-Based Apps
Visual Studio Magazine, June 23, 2005

Forms vs. Applications
Jakob Nielsen's Alertbox, Sept. 19, 2005

Beyond Browsing: Syntek and Groove Networks get P2P ready to go to work
James Karney, InternetWorld
September, 2001

Internet Perspective
Bruce Tognazzini, AskTog.com
May, 2001

www.p2p.edu: Rip, Mix & Burn Your Education
by Thom Gillespie, Maître d'Igital

Ready for Web Apps?
Jim Seymour, PC Magazine
October, 1999

Summer, 2001

The Death Of The Web Is Inevitable, According To Forrester Research
Mariko Zapf, Forrester Research, Inc.
May, 2001

ZDNet Special Report: Instant Messaging
ZDNet News

Instant Messaging Goes to Work
David Legard, IDG News Service
November, 2000

Web services a savior?
Bob Lewis, InfoWorld
August, 2001

SirCam worm still a serious threat
Robert Lemos, CNET News.com
September, 2001

Stubborn AOL may have last laugh
Jim Hu, CNET News.com
June, 2001

The misery of Web applications
D.F. Tweeney, The Tweeney Report
March 2000

Freenet developer to create commercial apps
Ed Scannell, InfoWorld
April, 2001

The Death of TCP/IP
Why the Age of Internet Innocence is Over
Robert X. Cringely
August, 2001

Response Times: The Three Important Limits
Jakob Nielson, excerpt from Ch. 5 of his book Usability Engineering

The Case Against Strained Carrots
Scott Raney, MetaCard Corporation
1996

The Network is the User Experience: Microsoft's .NET Announcement
Jakob Nielsen, August, 2001

My View: X Internet
George F. Colony, Chairman of the Board and CEO, Forrester
October 2000

Why 'Push' is now ripe (again)
Charles Cooper, ZDNet News
February, 2001

SETI and Distributed Computing
Garrett Moritz
1998

Web services on trial
Bob Lewis, InfoWorld
August, 2001

Code Red cost estimated at $2.6 billion
Reuters
August, 2001

P2P and XML in Business
Brian Buehling
July, 2001

Web services
Tom Sullivan, InfoWorld
March, 2001

Groove ships p-to-p platform
Ed Scannell, InfoWorld
April, 2001

Internet Winter
Why Internet Security is an Oxymoron
Robert X. Cringely
July, 2001

Scripting: Higher Level Programming for the 21st Century
J. Osterhaut, Scriptics
March, 1998

Drop-Down Menus: Use Sparingly
Jakob Nielsen, Alertbox
November, 2000

Runtime Revolution Embassy
Fourth World's repository of Revolution resources, and home of RevNet.

Web Sites

The Free Network Project

.Net Dev Center

Gnutella

OSF Distributed Computing Environment

Peer-to-Peer Working Group

Wired's Guide to Global File-Sharing

Napster

Gnutella Clients

Jabber

ICQ

AOL Instant Messenger

OpenP2P.com

OreillyNet

MacOrchard

Distributed Computing Projects

InfoAnarchy

Peertal

SETI@Home

Distributed.Net

MSN Messenger

Yahoo! Messenger

ACM Special Interest Group for Computer-Human Interaction

Interface Guidelines

Microsoft Windows Human Interface Guidelines

Motif Style Guide

KDE User Interface Guidelines

Apple Mac OS X Aqua Human Interface Guidelines

Irix User Interface Guidelines

Java Look and Feel Design Guidelines

Footnote: Most of the assertions in this article are based solely on anecdotal evidence, but not for lack of trying: there seems to be a dearth of published research of the usability differences between the Web and the Desktop experiences. I would appreciate the opportunity to update this paper. If you find interesting links about this sort of thing on the Web, please send the URL to NetApps2010@fourthworld.com

Contact: web2014@fourthworld.com