Web Focus Corner:

Intermediaries: Ways Of Exploiting New Technologies

A number of new technologies have emerged recently, such as HTML 4.0, CSS2.0, XML and HTTP 1.1. Other new web technologies appear to be waiting, just around the corner. But we can't expect all users to upgrade to the latest version of their favourite browser. In this article **Brian Kelly** describes ways in which new technologies can be deployed.

Since February 1998 HTML 4.0 [1], CSS 2.0[2], the Mathematical Markup Language MathML [3] and the Extensible Markup Language XML [4] have all become **W3C Recommendations**. These web protocols, all of which are concerned with the way in which information can be represented and displayed, were initially **Working Drafts** which were developed by the appropriate W3C Working Group. The Working Drafts were then made publicly available as **W3C Proposed Recommendations**. Following a review period the Proposed Recommendations were voted on by W3C member organisations. Following a successful vote, the Proposed Recommendations became Recommendations.

But how widely used are these new web protocols? Today's web community, being so large and consisting of a majority who probably have little interest in the underlying web technologies, is reluctant to upgrade to new versions of browsers, due to a lack of knowledge, interest, technical skills or resources. More experienced web users remember the "browser wars" and the problems of being an early adopter of technologies.

What are the implications of this conservatism? Can we simply ignore these new technologies? Revisiting the recommendations and looking at the new features they provide it would appear that we cannot afford to ignore these technologies if we wish to develop richer, more elegant and more maintainable web services.

Conservatism in adopting new technologies is probably due to the following factors:

1. Lack of tools to create resources using the new technologies.
2. Time and resources needed to purchase and master tools once they become available.
3. Low uptake of browsers which support the new technologies and concerns over the interworking of new technologies with existing systems, such as old versions of browsers.

The lack of tools and the costs of deploying new tools will be addressed by the marketplace. But how are backwards compatibility issues to be addressed? This article considers this point.

Current Browser Usage Trends

Jakob Nielsen's *Alertbox* article published in March 22, 1998 [5] gave an analysis of browser usage. The results indicated that every week, about 2% of the users switched from previous releases to the new release. Moving most of the users to the new version would take a year (about 50 weeks at about 2% per week).

Figure 1 shows the changes in the profile of browsers accessing a national UK web service between July 1997 and June 1998. Although there has been a steady growth in the number of users using version 4 of a browser in that period, there are still only 30% of the hits coming from the latest versions of the browsers.

Browser	**July 1997**	**Sept 1997**	**Oct 1997**	**Nov 1997**	**Jan 1998**	**Mar 1998**	**May 1998**	**Jun 1998**
ie 2	0.6	0.5	0.6	0.6	0.4	0.6	0.3	0.3
ie 3	6.6	6.3	5.6	5.9	7.0	6.5	6.9	5.6
ie 4	0.3	0.6	1.2	1.3	2.5	2.3	4.3	6.4
ns 0/1	7.1	5.2	4.5	5.0	4.7	2.6	2.0	2.0
ns 2	26.6	22.3	21.4	19.0	18.0	18.1	14.8	10.8
ns 3	53.5	54.6	58.4	58.0	53.0	55.3	51.8	49.9
ns 4	4.2	9.31	7.8	9.0	13.0	14.0	19.4	24.2

Figure 1a - User Agents Accessing A National UK Web Service

Although, of course, these figures are open to interpretation (they indicate "sessions" and not users, for example) they do show that a significant minority of users are still using "old" browsers (version 2 and earlier) and less than a third are using the latest generation.

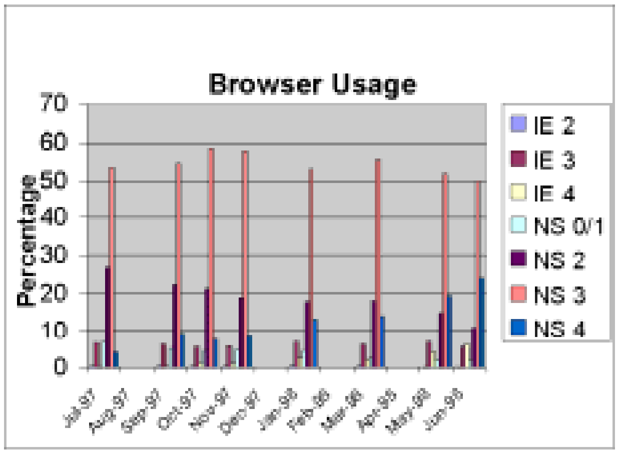


Figure 1b Browser Usage in Tabular Format

Web Deployment Decisions

Given the range of versions of browsers in use and the slow rate of upgrade to newer versions, how can a web developer exploit new technologies which may enable the developer's service to add useful new features, better and richer interfaces or improved performance? Possible solutions include:

1. Do nothing
2. Move to new technologies
3. Deploy new technologies in a backwards-compatible way
4. Provide end users with options for choosing between technologies
5. Deploy technologies on web server
6. Deploy technologies on client
7. Make use of *content negotiation*
8. Make use of *user-agent negotiation*

Let's look at these options in some more detail.

1. Do Nothing

Perhaps the simplest option, which can be a deliberate choice or the default choice through inertia, is to continue with existing working practices. This may be the cheapest option (in the short term at least) since no new tools are needed, there are no new training requirements and a consistent service is provided for the end user. On the other hand this decision may simply delay a transition, and add to the maintenance work when the transition eventually takes place.

2. Move to New Technologies

The opposite to doing nothing is to move completely to the use of new technologies. This approach may be applicable for developing an Intranet service in which the developer has knowledge of and control over the local systems or for providing a proof-of-concept. However providers of mainstream services are unlikely to wish to adopt this approach.

3. Deploy New Technologies in a Backwards-Compatible Way

New technologies may be deployed in a backwards compatible way. Web protocols are intended to be backwards compatible, wherever possible, so that, for example, new HTML elements and attributes are ignored by older browsers. In this article, for example, inline style sheets have been used to change the colour of the heading of this paragraph. If you are using a browser which support style sheets (such as version 4 of Netscape or Internet Explorer) the heading will be displayed in brown, whereas other browsers with display the headings in the default colour.

Unfortunately due to bugs in the browsers (if you believe in the cock-up theory) and inconsistencies in the ways in which Netscape and Microsoft have interpreted new technologies (if you're a believer in conspiracy theories) not all new technologies are implemented consistently or degrade gracefully. It will, unfortunately, be necessary to test how new technologies work on a variety of systems - which may not be easy if, for example, you do not have access to Netscape on a Macintosh or Internet Explorer on a Unix workstation.

4. Provide End Users with Options for Choosing Between Technologies

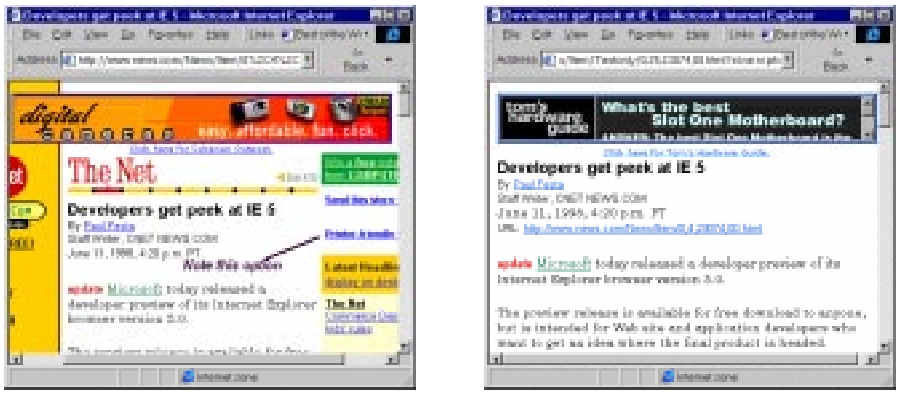
A third alternative is to provide the end user with a choice of options. We have probably all seen pages containing links such as "*Click here for a non-frames version*", or "*Click here if you have the Macromedia Plugin*". This option is not particularly suited to access by naive users who may not know what functions their browser supports or what a plugin is. In addition the information provider will have multiple versions of the resources to maintain. This can be expensive, although some authoring systems, such as **NetObjects Fusion** [6] can maintain multiple versions of resources automatically.

5. Deploy Technologies on Web Server

Technologies can be deployed on web servers and made seamlessly available to the client. For example the following button can be used to validate this document using an HTML validation service running in the US.

Validate this page

Although we are familiar with surfing the web to access *information* from a variety of servers, the notion of using third party *applications* does not appear to have taken off with the UK Higher Education community. However we could envisage a model in which, say, an institutional or even national *intermediary service* transformed a richly structured resource (using, say, XML) into a resource which can be processed by the browser on the user's desktop. The intermediary service could be invoked in a variety of ways, such as by configuring the browser appropriately (e.g. use of the resource's MIME type or using the client's autoproxy function) or perhaps in some cases through end user action. For example see the ("*Printer Friendly version*" option which can be used to reformatting a framed page to a form suitable for printing at the C|Net web site (**`<URL: http://www.news.com/News/Item/0%2C4%2C23074%2C00.html?dd.ne.tx.ts.>`**).


Developers get peek at IE 5 - Microsoft Internet Explorer
The Net
Note this option
Printer-friendly
Developers get peek at IE 5

Developers get peek at IE 5
By Paul Festa
Staff Writer, CNET NEWS.COM
June 11, 1998, 4:20 p.m. PT

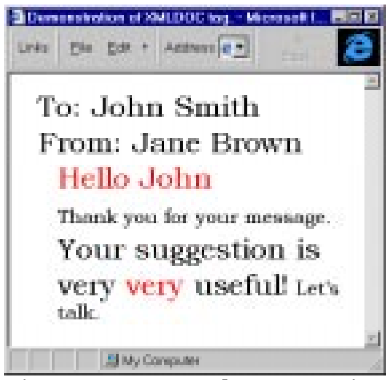
update Microsoft today released a developer preview of its Internet Explorer browser version 5.0.

The preview release is available for free download to anyone, but is intended for Web site and application developers who want to get an idea where the final product is headed.

Figure 2 Web Resource Which Can be Reformatted for Printing

6. Deploy Technologies on Client

Rather than deploying new technologies on the server, they could be deployed within the browser by using, for example, a scripting language on the browser. Such techniques were mentioned in the *What Is XML?* article in Ariadne edition 15 [7] which included an example of how XML document could be displayed in a browser which did not support XML by using a script to convert from XML to HTML, as shown in Figure 3.



To: John Smith
From: Jane Brown
Hello John
Thank you for your message.
Your suggestion is very very useful! Let's talk.

Figure 3 XML Deployment Using JavaScript

7. Make Use of *Content Negotiation*

The elegant way of deploying new formats which was originally envisaged by developers of web protocols was through *transparent content negotiation* (TCN) [8]. It was originally intended that browsers would send a list of the file formats that it could accept and that if multiple formats of the same resource were available the server would sent the most suitable (according to some algorithm, such as the smallest file size).

The World Wide Web Consortium make use of TCN on their home page. As shown in Figure 4 if you access their home page with an old browser which does not support the PNG graphical format you will be sent a GIF image. If, on the other hand, you are using a browser which does support PNG, such as Netscape 4 or Internet Explorer 4, you will be sent the PNG file.

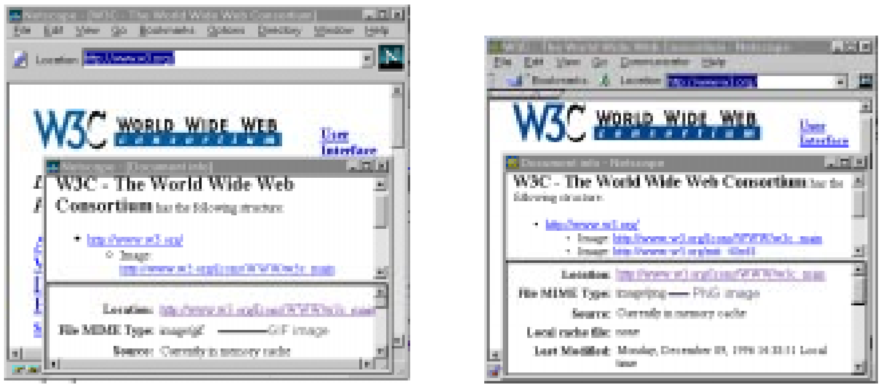


Figure 4 Netscape 3 shows a GIF logo but Netscape 4 shows a PNG version

Unfortunately transparent content negotiation does not appear to be widely deployed, in part due to lack of awareness of the feature, but also due to the lack of support available in authoring tools and the poor implementation of TCN in browsers.

Although Holtman has proposed [9] that *content feature negotiation* could enable new features, such as new HTML elements, to be deployed in a way in which clients and servers could transparently negotiate over use of such features, it is by no means certain that this proposal will become standardised within the IETF or implemented by the software vendors.

8. Make Use of *User Agent Negotiation*

A less elegant way of deploying new technologies is through *User Agent Negotiation*. Rather than browsers sending a list of formats it supports, this requires web administrators to have a knowledge of formats and technologies supported by different browsers and by different versions of the same browser. Some large web sites make use of this technique, as illustrated in Figure 5 which shows that the different interfaces to AltaVista [10] presented to Internet Explorer 4 and NCSA Mosaic.

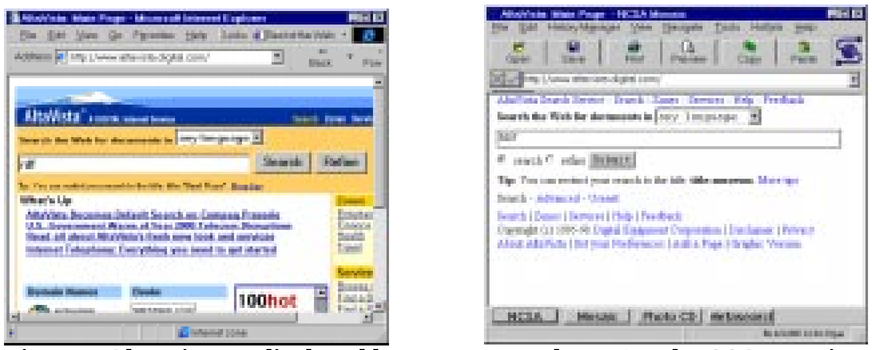


Figure 5 Alta Vista as displayed by Internet Explorer 4 and NCSA Mosaic

In this example rich content, including use of tables, images and active maps, is sent to Internet Explorer, whereas NCSA Mosaic is sent very simple content.

User agent negotiation has a number of advantages. For example complex client side scripts to check the functionality of the browser are not needed. Unlike the policy of requiring the end user to make use of a specified browser (e.g. using the "*Best viewed in Netscape*" icon) the content is designed for the browser the end user is using. Indeed the page could legitimately contain an icon saying "*Best viewed in **your** browser*".

Critics of user agent negotiation argue that the maintenance of different pages for not only different browsers, but also different versions of the same browser together with corresponding feature lists - and possibly bug lists - is likely to be difficult. In practice, however, it is unlikely that most service providers would want to support every version of a browser. In addition the latest generation of document management systems, such as PHP [11], Microsoft's Active Server Pages [12] and Vignette's StoryServer [13], provide server-side scripting capabilities which can automate the management of customisable web pages.

An example of user-agent negotiation is hosted by the World Wide Web Consortium. A series of core style sheets are available from <URL: http://www.w3.org/StyleSheets/Core/>. As described in the development interface page [14], the styleserver service makes use of "browser sniffing" - its name for user agent negotiation. This service omits certain style sheet options from particular browsers in order to avoid certain known implementation problems.

The systems mentioned above can create what are known as *dynamic* pages. Although the pages may look static if they are created based on the browser the user is using, they may be created dynamically. This approach *may* cause problems with caching the resource, since dynamically-generated content can either be marked as uncacheable, or marked for immediate expiry, or marked for expiry after an hour, a day, or whatever. This caching problem can, in principle, be overcome if the server management system enables cache times to be set and if the HTTP/1.1 protocol features to allow caching of negotiated responses is properly implemented.

Intermediaries

A paper on *Intermediaries: New Places For Producing and Manipulating Web Content* [15] presented at the WWW 7 conference proposed that, rather than letting web servers produce web content, *intermediaries*, processing systems which are located between the web server and browser, can provide new places for producing and manipulating Web data. The paper describes a programmable proxy server known as WBI (Web Browser Intelligence) [16] which stands between your browser and the web and enables intermediary applications to be developed. Sample WBI applications include sophisticated history management and notification systems. For example WBI can annotate web pages to highlight text that might be of particular interest or to add hyperlinks to related pages that you or others have seen.

A number of other examples of intermediaries were given in the *What Is XML?* article in Ariadne edition 15 [7] which described how XML support could be provided by intermediaries running on the client such as Javascript, ActiveX or Java applications. In the June 1998 edition of D-Lib [17] Andy Powell describes how URNs (Uniform Resource Names) based on DOIs (Document Object Identifiers) can be deployed using an intermediary based on the Squid proxy caching server.

Figure 6 illustrates use of this approach, by which Netscape can access a resource which has the URN **`urn:doi:10.1000/1`**. Although Netscape cannot normally resolve an address of this form, by configuring the Netscape autoproxy appropriately the URN can be forwarded to the Squid system which can resolve the address.

A similar approach to the use of intermediaries has been taken by the Open Journal eLib project [18] based at the University of Southampton. In this project a *Distributed Link Service* provides an external means of storing hyperlink information. The link data is merged with the document when the document is viewed, as illustrated in Figure 7.

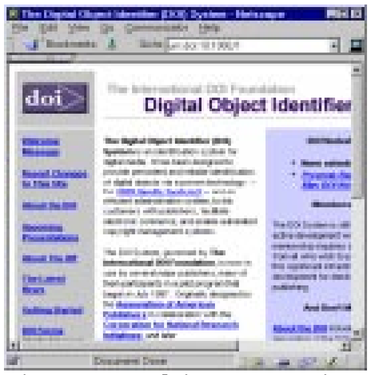
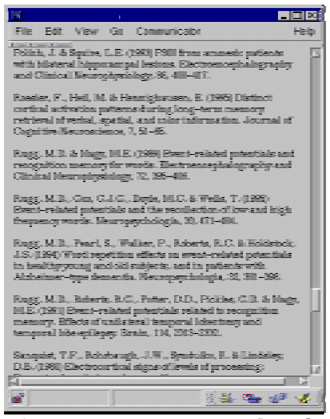

The Digital Object Identifier (DOI) System - Netscape
doi
Digital Object Identifier
Document: Done

Figure 6 Resolving DOIs Using an Intermediary



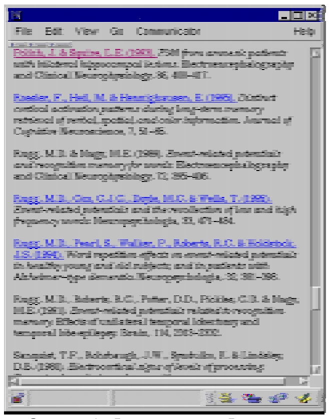


Figure 7 Open Journal Before and After Links Merged

The Future

What is the future for the deployment of new web technologies? Some argue that new technologies will be so clearly superior that users will discard their old browsers an upgrade in a short period of time and give the example of the rapid migration from Gopher clients to Web browsers. However the resources needed to upgrade browser software - especially if it has been installed on individuals PCs rather than on a server - may, unfortunately, rule out this option.

Rather than ruling out deployment of new technologies completely, content negotiation, user agent negotiation and intermediaries may enable new technologies to be deployed while maintaining compatibility with existing browser technologies.

Figure 8 illustrates a model for the deployment of new technologies.

In Figure 8 it is envisaged that data will be stored as HTML or XML resources or in a backend database. The backend server scripting interface can process the data, such as converting the file format or customising the output. A proxy server, implemented as a national service or within an institution, a desktop proxy, or a browser extension (implemented in, say, JavaScript or Java) could then provide functionality not provided natively by the browser or server.

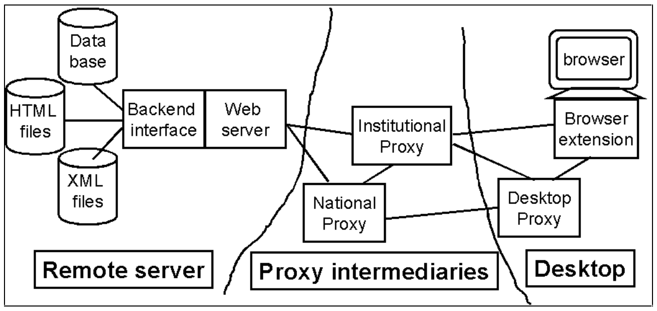


Figure 8 A Model for the Future

Some scenarios for the use of intermediaries are given below:

Deploying URNs
An institutional proxy service resolves URNs of popular resources into the URL of a "nearby" mirror of the resource.

XML Deployment
A client-side intermediary, implemented in JavaScript, converts XML resources into HTML.

Extended Browser Functionality
An desktop proxy service updates the view of documents it receives, so that broken links are flagged, and an indication is given of resources which have recently changed.

References

1. HTML 4.0 Specification, W3C, <URL: http://www.w3.org/TR/REC-html40>
2. Cascading Style Sheets, Level 2 (CSS2) Specification, W3C, <URL: http://www.w3.org/TR/REC-CSS2>
3. Mathematical Markup Language (MathML) 1.0 Specification, W3C, <URL: http://www.w3.org/TR/REC-MathML>
4. Extensible Markup Language (XML) 1.0 Specification, W3C, <URL: http://www.w3.org/TR/REC-xml>
5. The Increasing Conservatism of Web Users, Alertbox, <URL: http://www.useit.com/alertbox/980322.html>
6. NetObjects Fusion, NetObjects, <URL: http://www.netobjects.com/>
7. What is XML, Ariadne Issue 15, <URL: http://www.ariadne.ac.uk/issue15/what-is/>
8. Transparent Content Negotiation in HTTP, Koen Holtman, <URL: http://gewis.win.tue.nl/~koen/conneg/>
9. Content Feature Tag Registration Procedure, Koen Holtman, <URL: http://www.imc.org/draft-ietf-conneg-feature-reg>
10. Alta Vista, <URL: http://www.altavista.digital.com/>
11. PHP, <URL: http://www.php.net/>
12. ASP, Microsoft, <URL: http://www.microsoft.com/NTServer/Basics/WebServices/Features/FeatContMgmt.asp>
13. StoryServer, Vignette, <URL: http://www.vignette.com/Products/0,1056,0,00.html>
14. StyleServer Development Interface, Verso Inc., <URL: http://style.verso.com/styleserver/>
15. Intermediaries: New Places For Producing and Manipulating Web Content, <URL: http://www7.conf.au/programme/fullpapers/1895/com1895.htm>
16. WBI, IBM, <URL: http://wwwcssrv.almaden.ibm.com/wbi/>
17. Resolving DOI Based URNs Using Squid, D-Lib, June 1998, <URL: http://mirrored.ukoln.ac.uk/lis-journals/dlib/dlib/dlib/june98/06powell.html>
18. Open Journal Project, <URL: http://journals.ecs.soton.ac.uk/>

This article was originally published in *the Web Focus Corner* column in the web version of Ariadne, issue 15. See <URL: `http://www.ariadne.ac.uk/issue16/web-focus/`>