Using Ajax to Empower Dynamic Searching

Judith Wusteman and
Pádraig O'hIceadha

The use of Ajax, or Asynchronous JavaScript + XML, can result in Web applications that demonstrate the flexibility, responsiveness, and usability traditionally found only in desktop software. To illustrate this, a repository metasearch user interface, OJAX, has been developed. OJAX is simple, unintimidating but powerful. It attempts to minimize upfront user investment and provide immediate dynamic feedback, thus encouraging experimentation and enabling enactive learning.

This article introduces the Ajax approach to the development of interactive Web applications and discusses its implications. It then describes the OJAX user interface and illustrates how it can transform the user experience.

With the introduction of the Ajax development paradigm, the dynamism and richness of desktop applications become feasible for Web-based applications. OJAX, a repository metasearch user interface, has been developed to illustrate the potential impact of Ajax-empowered systems on the future of library software.[1]

This article describes the Ajax method, highlights some uses of Ajax technology, and discusses the implications for Web applications. It goes on to illustrate the user experience offered by the OJAX interface.

Ajax

In February 2005, the term Ajax acquired an additional meaning: Asynchronous JavaScript + XML.[2] The concept behind this new meaning, however, has existed in various forms for several years. Ajax is not a single technology but a general approach to the development of interactive Web applications. As the name implies, it describes the use of JavaScript and XML to enable asynchronous communication between browser clients and server-side systems.

As explained by Garrett, the classic Web application model involves user actions triggering a hypertext transfer protocol (HTTP) request to a Web server.[3] The latter processes the request and returns an entire hypertext markup language (HTML) page. Every time the client makes a request to the server, it must wait for a response, thus potentially delaying the user. This is particularly true for large data sets. But research demonstrates that response times of less than one second are required when moving between pages if unhindered navigation is to be facilitated through an information space.[4]

The aim of Ajax is to avoid this wait. The user loads not only a Web page, but also an Ajax engine written in JavaScript. Users interact with this engine in the same way that they would with an HTML page, except that instead of every action resulting in an HTTP request for an entire new page, user actions generate JavaScript calls to the Ajax engine. If the engine needs data from the server, it requests this asynchronously in the background. Thus, rather than requiring the whole page to be refreshed, the JavaScript can make rapid incremental updates to any element of the user interface via brief requests to the server. This means that the traditional page-based model used by Web applications can be abandoned; hence, the pacing of user interaction with the client becomes independent of the interaction between client and server.

XMLHttpRequest is a collection of application programming interfaces (APIs) that use HTTP and JavaScript to enable transfer of data between Web servers and Web applications.[5] Initially developed by Microsoft, XMLHttpRequest has become a de facto standard for JavaScript data retrieval and is implemented in most modern browsers. It is commonly used in the Ajax paradigm. The data accessed from the HTTP server is usually in Extensible Markup Language (XML) but another format, such as JavaScript Object Notation, could be used.[6]

Applications of Ajax

Google is the most significant user of Ajax technology to date. Most of its recent innovations, including Gmail, Google Suggest, Google Groups, and Google Maps, employ the paradigm.[7]

The use of Ajax in Google Suggest improves the traditional Google interface by offering real-time suggestions as the user enters a term in the search field. For example, if the user enters xm, Google Suggest might offer refinements such as xm radio, xml, and xmods. Experimental Ajax-based auto-completion features are appearing in a range of software.[8] Shanahan has applied the same ideas to the Amazon online bookshop.[9] His experimental site, Zuggest, extends the concept of auto-completion: as the user enters a term, the system automatically triggers a search without the need to hit a search button.

The potential of Ajax to improve the responsiveness and richness of library applications has not been lost on the library community.[10] Several interesting experiments have been tried. At OCLC, for example, a "suggest-like service," based on controlled headings from the world-

Judith Wusteman (judith.wusteman@ucd.ie) is a lecturer in the UCD School of Information and Library Studies, University College Dublin, Ireland.

wide union catalog, WorldCat, has been implemented.[11] Ajax has also been used in the OCLC DeweyBrowser.[12] The main page of this browser includes four iframes, or inline frames, three for the three levels of Dewey Decimal Classification and a fourth for record display.[13] The use of Ajax allows information in each iframe to be updated independently without having to reload the entire page.

Implications of Ajax

There have been many attempts to enable asynchronous background transactions with a server. Among alternatives to Ajax are Flash, Java Applets, and the new breed of XML user-interface language formats such as XML User Interface Language (XUL) and Extensible Application Markup Language (XAML).[14] These all have their place, particularly languages such as XUL. The latter is ideal for use in Mozilla extensions, for example. Combinations of the above can and are being used together; XUL and Ajax are both used in the Firefox extension version of Google Suggest.[15] The main advantage of Ajax over these alternative approaches is that it is nonproprietary and is supported by any browser that supports JavaScript and XMLHttpRequest—hence, by any modern browser.

It could be validly argued that complex client-side JavaScript is not ideal. In addition to the errors to which complex scripting can be prone, there are accessibility issues. Best practice requires that JavaScript interaction adds to the basic functionality of Web-based content that must remain accessible and usable without the JavaScript.[16] An alternative non-JavaScript interface to Gmail was recently implemented to deal with just this issue.

A move away from scripting would, in theory, be a positive step for the Web. In practice, however, procedural approaches continue to be more popular; attempts to supplant them, as epitomized by XHTML 2.0, simply alienate developers.[17]

It might be assumed that the use of Ajax technology would result in a heavier network load due to an increase in the number of requests made to the server. This is a misconception in most cases. Indeed, Ajax can dramatically reduce the network load of Web applications, as it enables them to separate data from the graphical user interface (GUI) used to display it. For example, each results page presented by a traditional search engine delivers, not only the results data, but also the HTML required to render the GUI for that page. An Ajax application could deliver the GUI just once and, after that, deliver data only. This would also be possible via the careful use of frames; the latter could be regarded as an Ajax-style technology but without all of Ajax's advantages.

From client-server to SOA

The dominant model for building network applications is the client/server approach, in which client software is installed as a desktop application and data generally reside on a server, usually in a database.[18] This can work well in a homogenous single-site computing environment. But institutions and consortia are likely to be heterogeneous and geographically distributed. PCs, Macs, and cell phones will all need access to the applications, and Linux may require support alongside Windows. Even if an organization standardizes solely on Windows, different versions of the latter will have to be supported, as will multiple versions of those ubiquitous Dynamic Link Libraries (DLLs). Indeed, the problems of obtaining and managing conflicting DLLs have spawned the term "DLL hell."[19]

In Web applications, a standard client, the browser, is installed on the desktop but most of the logic, as well as the data, reside on the server. Of course, the browser developers still have to worry about "DLL hell," but this need not concern the rest of us.

"Speed must be the overriding design criterion" for Web pages.[20] But the interactivity and response times possible with client/server applications are still not available to traditional Web applications. This is where Ajax comes in: it offers, to date, the best of the Web application and client/server worlds. Much of the activity is moved back to the desktop via client-side code. But the advantages of Web applications are not lost: the browser is still the standard client.

Service-Oriented Architecture (SOA) is an increasingly popular approach to the delivery of applications to heterogeneous computing environments and geographically dispersed user populations.[21] SOA refers to the move away from monolithic applications toward smaller, reusable services with discrete functionality. Such services can be combined and recombined to deliver different applications to users. Web Services is an implementation of SOA principles.[22] The term describes the use of technologies such as XML to enable the seamless interoperability of Web-based applications. Ajax enables Web Services and hence enables SOA principles. Thus, the adoption of Ajax facilitates the move toward SOA and all the advantages of reuse and integration that this offers.

ARC

ARC is an experimental open-source metasearch package available for download from the SourceForge open-source foundry.[23] It can be configured to harvest Open

Archives Initiative-Protocol for Metadata Harvesting (OAI-PMH)-compliant data from multiple repositories.[24] The harvested results are stored in a relational database and can be searched using basic Web forms. ARC's Advanced Search form is illustrated in figure 1.

Applying Ajax to the search GUI

The use of Ajax has the potential to narrow the gulf between the responsiveness of GUIs for Web applications and those for desktop applications. The flexibility, usability, and richness of the latter are now possible for the former. The OJAX GUI, illustrated in figure 2, has been developed to demonstrate how Ajax can improve the richness of ARC-like GUIs. OJAX, including full source code, is available under the open-source Apache license and is hosted on SourceForge.[25]

OJAX comprises a client-side GUI, implemented in JavaScript and HTML, and server-side metasearch Web Services, implemented in Java. The Web Services connect directly to a metasearch database created by ARC from harvested repositories. The database connectivity leverages several libraries from the Apache Jakarta project, which provides open-source Java solutions.[26]

Development process

The OJAX GUI was developed iteratively using Agile software development methods.[27] Features were added incrementally and feedback gained from a proxy user. In order to gain an in-depth understanding of the system and the implications for the remainder of the GUI, features were initially built from scratch, using object-oriented JavaScript.They were then rebuilt using three open-source JavaScript libraries: Prototype, script.aculo .us, and Rico.[28]

Prototype provides base Ajax capability. It also includes advanced functionality for object-oriented JavaScript, such as multiple inheritance. The other two libraries are built on top of Prototype. The script.aculo. us library specializes in dynamic effects, such as those used in auto-completion. The Rico library, developed by Sabre, provides other key JavaScript effects—for example, dynamic scrollable areas and dynamic sorting.[29]

Storyboard

One of the aims of the National Information Standards Organization (NISO) Metasearch Initiative is to enable all library users to "enjoy the same easy searching found in web-based services like Google."[30] Adopting this approach, OJAX incorporates the increasingly common concept of the search bar, popularized by the Google Toolbar.[31] OJAX aims to be as simple, uncluttered, and unthreatening as possible. The goal is to reflect the simple-search experience while, at the same time, providing the power of an advanced search. Thus, the user interface has been kept as simple as possible while maintaining equivalent functionality with the ARC Advanced Search interface. All ARC functionality, with the exception of the grouping feature, is provided.

To help the intuitive flow of the operation, the fields are set out as a sentence:

Find **[term(s)]** in **[all archives]** from **[earliest year]** until **[this year]** in **[all subjects]**

Tool tips are available for text-entry fields. By default, searching is on author, title, and abstract. These fields map to the creator, title, and description Dublin Core metadata fields harvested from the original repositories.[32] The search can be restricted by deselecting unwanted fields.

ARC supports both MySQL and Oracle databases.[33] MySQL has been chosen for OJAX as MySQL is an open-source database. Boolean search syntax has been

Figure 1. ARC's Advanced Search form

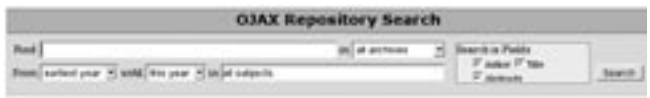

Figure 2. The OJAX Metasearch User Interface

implemented in OJAX to allow for more powerful searching. The syntax is similar to that used by Google in that it identifies AND/OR and *exact phrase* functionality by +/- and " ". Hence it preserves the user's familiarity with basic Google search syntax. However, it is not as powerful as the full Google search syntax; for example, it does not support query modifiers such as:

intitle: [34]

The focus of this research is the application of Ajax to the search GUI and not the optimization of the power or expressive capability of the underlying search engine. However, the implementation of an alternative back end that uses a full-text search engine, such as Apache Lucene, would improve the expressive power of advanced queries.[35] Full-text search expressiveness is likely to be key to the usability of OJAX, ensuring its adequacy for the advanced user without alienating the novice.

Unifying the user interface

One of the main aims of OJAX is the unification of the user interface. Instead of offering distinct options for simple and advanced search and for refining a completed search, the interface is sufficiently dynamic to make this unnecessary. The user need never navigate between pages because all options, both simple and advanced, are available from the same page. And all results are made available on that same page in the form of a scrollable list. The only point at which a new page is presented is when the resource identifier of a result is clicked. At this stage, a pop-up window, external to the OJAX session, displays the full metadata for that resource. This page is generated by the external repository from which the record was originally harvested.

Simple and advanced search options are usually kept separate because most users are unwilling or unable to use the latter.[36] Furthermore, the design of existing search-user interfaces is based on the assumption that the retrieval of results will be sufficiently time-consuming that users will want to have selected all options beforehand.

With OJAX, however, users do not have to make a complete choice of all the options they might want to try before they see any results. As data are entered, answers flow to accommodate them. Because the interface is so dynamic and responsive and because users are given immediate feedback, they do not have to be concerned about wasting time due to the wrong choice of search options. Users iterate toward the search results they require by manipulating the results in real time. The reduced level of investment that users must make before they achieve any return from the system should encourage them to experiment, hence promoting enactive learning.

Auto-completion

In order to provide instant feedback to the user, the search-terms field and the subject field use Ajax to autocomplete user entries. Figure 3 illustrates the result of typing *Smith* in the search-terms field. A list is automatically dropped down that itemizes all matches and the number of their occurrences. Users select the term they want, the entire field is automatically completed, and a search is triggered.

The ARC system denormalizes some of the harvested data before saving them in its database. For example, it merges all the author fields into one single field, each name separated by a bar character. To enable the OJAX auto-completion feature, it was necessary to renormalize the names. A new table is used to store each name in a separate row; names are referenced by the resource identifier. To enable this, ARC's indexing code was updated so that it creates this table as it indexes records extracted from the OAI-PMH feed.

In its initial implementation, OJAX uses a simple algorithm for auto-completion. Future work will involve developing a more complex heuristic that will return results more closely satisfying user requirements.

Auto-search

As already mentioned, a central theme of OJAX is the attempt to reduce the commitment necessary from users before they receive feedback on their actions. One way in which dynamic feedback is provided is the triggering of an immediate search whenever an entire option has been selected. Examples of entire options include choice of an archive or year and acceptance of a suggested autocompletion. In addition, the following heuristics are used to identify when a user is likely to have finished entering a search term and, thus, when a search should be triggered:

1. Entering a space character in the search-terms field or subject field
2. Tabbing out of a field after having modified its contents
3. Five seconds of user inactivity for a modified field

The third heuristic aims to catch some of the edge cases that the other heuristics may miss. It is assumed likely that a term has been completed if a user has made no edits in the last five seconds. As each term will be

separated by a space, it is only the last term in a search phrase that is likely not to trigger an auto-search via the first heuristic.

Users can click the search button whenever they wish, but they should never *have* to click it. The Zuggest system abandons the search button entirely; OJAX retains it, mainly in order to avoid confounding user expectations.[37]

While a search is in progress, the search button is greyed out and acquires a red border. This is particularly useful in alerting the user that a search has been automatically triggered.

This is the only feature of OJAX that may have an impact on network load in terms of slightly higher traffic. However, the increased number of requests is offset by a reduction in the size of each response because the GUI is not downloaded with it. For example, initiating a search in ARC results in an average response size of 57.32K. The response is in the form of a complete HTML page. Initiating a search in OJAX results in an average response size of 7.96K. The latter comprises a Web Service response in XML. In other words, more than seven OJAX auto-searches would have to be triggered before the size of the initial search result in ARC was exceeded.

Dynamic archive list

The use of Ajax enables a static HTML page to contain a small component of dynamic data without the entire page having to be dynamically generated on the server. OJAX illustrates this: the contents of the drop-down box listing the searchable archives are not hard-coded in the HTML page. Rather, when the page is loaded, an Ajax request for the set of available archives is generated. This is a useful technique; static HTML pages can be cached by browsers and proxy servers, and only the dynamic portion of the data, perhaps those used to personalize the page, need be downloaded at the start of a new session.

Dynamic scrolling

Searches commonly produce thousands of results. Typical systems, such as Google and ARC, make these results available via a succession of separate pages, thus requiring users to navigate between them. Finding information by navigating multiple pages can take longer than scrolling down a single page, and users rarely look beyond the second page of search results.[38] To avoid these problems and to encourage users to look at more of the available results, those results could be made available in one scrollable list. But, in a typical non-Ajax application, accessing a scrollable list of, say, two thousand items would require the entire list to be downloaded via one enormous HTML page. This would be a huge operation; if it did not crash the browser, it would, at least, result in a substantial wait for the user.

The Rico library provides a feature to enable dynamic scrollable areas. It uses Ajax to fetch more records from the server when the user begins to scroll off the visible area. This is used in the display of search results in OJAX, as illustrated in figure 4. To the user, it appears that the scrollable list is seamless and that all 4,678 search results are already downloaded. In fact, only 386 have been downloaded. The rest are available at the server. As the user scrolls further down, say to item 396, an Ajax request is made for the next ten items. Any item downloaded is cached by the Ajax engine and need not be requested again if, for example, the user scrolls back up the list.

A dynamic information panel is available to the right of the scroll bar. It shows the current scroll position in relation to the beginning and end of the results set. In

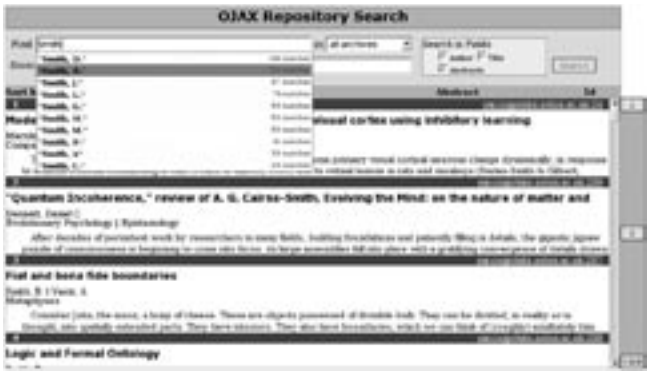

Figure 3. Auto-completion in the search terms field

Figure 4. Display of search results and dynamic information panel

figure 4, the information panel indicates that there are 4,678 results for this particular search and that the current scroll position is at result number 386. This number updates instantly during scrolling, preserving the illusion that all results have been downloaded and providing users with dynamic feedback on their progress through the results set. This means that users do not have to wait for the main results window to refresh to identify their current position.

Auto-expansion of results

OJAX aims to provide a compact display of key information, enabling users to see multiple results simultaneously. It also aims to provide simple access to full result details without requiring navigation to a new Web page.

In the initial results display, only one line each of the title, authors, and subject fields, and two lines of the abstract, are shown for each item. As the cursor is placed on the relevant field, the display expands to show any hidden detail in that field. At the same time, the background color of the field changes to blue. When the cursor is placed on the bar containing the resource identifier, all display fields for that item are expanded, as illustrated in figure 5.

This expansion is enabled via simple Cascading Style Sheet (CSS) features. For example, the following CSS declaration hides all but the first line of authors:

```
#searchResults td div
{
    overflow:hidden;
height: 1.1em
}
```

When the cursor is placed on the author details, the overflow becomes visible and the display field changes its dimensions to fit the text inside it:

```
#searchResults td div:hover
{
overflow:visible;
    height:auto
}
```

Sorting results

Another method used by OJAX to minimize upfront user investment is to provide initial search results before requiring the user to decide on sort options. Because results are available so quickly and because they can be re-sorted so rapidly, it is not necessary to offer pre-search selection of sort options. Ajax facilitates rapid presentation of results; after a re-sort, only those on the first screen must be downloaded before they can be presented to the user.

Results may be sorted by title, author, subject, abstract, and resource identifier. These options are listed on the gray bar immediately above the results list. Clicking one of these options sorts the results in ascending order; an upward-pointing arrow appears to the right of the Sort option chosen, as illustrated in figure 6. Clicking on the option again sorts in descending order and reverses the direction of the arrow. Clicking on the arrow removes the sort; the results revert to their original order.

Functionality for the Sort feature is provided by the Rico JavaScript library. Server-side implementation supports these features by caching search results so that it is not necessary to regenerate them via a database query each time.

Figure 5. Auto-expansion of all fields for item number 386

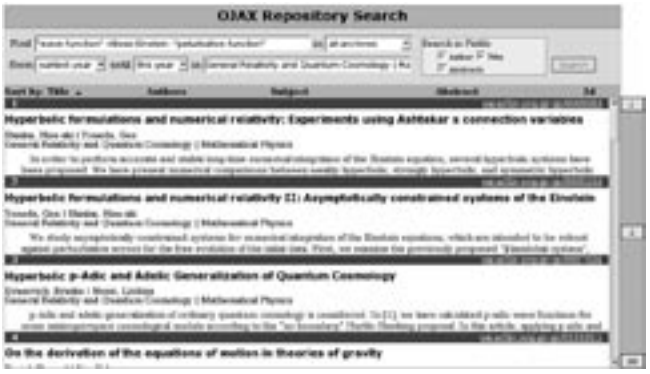

Figure 6. Results being sorted in ascending order by title

Search history

Several experimental systems—for example, Zuggest—have employed Ajax to facilitate a search-history feature. A similar feature could be provided for OJAX. A button could be added to the right of the results list. When chosen, it could expand a collapsible search-history sidebar. As the cursor was placed on one of the previous searches listed in the sidebar, a call out, that is, a speech bubble, could be displayed. This could provide further information such as the number of matches for that search and a summary of the search results clicked on by the user. Clicking one of the previous searches would restore those search results to the main results window.

This feature would take advantage of the Ajax persistent JavaScript engine to maintain the history. Its use could help counter concerns about Ajax technology "breaking" the Back button; the feature could be implemented so that the Back button returned the user to the previous entry in the search history.[39] In fact, this implementation of Back-button functionality could be more useful than the implementation in Google, where hitting the Back button is likely to take the user to an interim results page; for example, it might simply take the user from page 3 of results to page 2 of results.

Scrapbook

Users browsing through search results on OJAX would require some simple method of maintaining a record of those resource details that interested them. Ajax could enable the development of a useful scrapbook feature to which such resource details could be copied and stored in the persistent JavaScript engine. OJAX could further leverage a shared bookmark Web Service, such as del.icio.us or Furl, to save the scrapbook for use in future sessions and to share it with other members of a research or interest group.[40]

Potential developments for OJAX

As well as searching a database of harvested metadata, the OJAX user interface could also be used to search an OAI-PMH-compliant repository directly. With appropriate implementation, all of OJAX's current features could be made available, apart from auto-completion.

A recent development has enabled the direct indexing of repositories by Google using OAI-PMH.[41] The latter provides Google with additional metadata that can be searched via the Google Web Services APIs. The current OJAX Web Services could be replaced by the Google APIs, thus eliminating the need for OJAX to host any server-side components. Hence, OJAX could become an alternative GUI for Google searching.

Conclusion

OJAX demonstrates that the use of Ajax can enable features in Web applications that, until now, have been restricted to desktop applications. In OJAX, it facilitates a simple, nonthreatening, but powerful search user interface. Page navigation is eliminated; dynamic feedback and a low initial investment on the part of users encourage experimentation and enable enactive learning. The use of Ajax could similarly transform other Web applications aimed at library patrons.

However, Ajax is still maturing, and the barrier to entry for developers remains high. We are a long way from an Ajax button appearing in Dreamweaver. Reusable, well-tested components, such as Rico, and software frameworks, such as Ruby on Rails, Sun's J2EE framework, and Microsoft's Atlas, will help to make Ajax technology accessible to a wider range of developers.[42]

As with all new technologies, there is a temptation to use Ajax simply because it exists. As Ajax matures, it is important that its focus does not become the enabling of "cool" features but remains the optimization of the user experience.

References and notes

1. OJAX homepage, http://ojax.sourceforge.net (accessed Apr. 5, 2006).

2. J. J. Garrett, "Ajax: A New Approach to Web Applications," Feb. 18, 2005, www.adaptivepath.com/publications/essays/archives/000385.php (accessed Nov. 11, 2005).

3. Ibid.

4. J. Nielsen, "The Need for Speed," *Alertbox* Mar. 1, 1997, www.useit.com/alertbox/9703a.html (accessed Nov. 11, 2005).

5. Dynamic HTML and XML: The XMLHttpRequest Object, http://developer.apple.com/internet/webcontent/xmlhttpreq.html (accessed Apr. 5, 2006).

6. JavaScript Object Notation, Wikipedia definition, http://en.wikipedia.org/wiki/JSON (accessed Apr. 5, 2006).

7. Google Gmail, http://mail.google.com (accessed Apr. 5, 2006); Google Suggest, www.google.com/webhp?complete=1&hl=en (accessed Apr. 5, 2006); Google Groups, http://groups.google.com (accessed Apr. 5, 2006); Google Maps, http://maps.google.com (accessed Apr. 5, 2006).

8. P. Binkley, "Ajax and Auto-completion," *Quædam cuiusdam blog* May 18, 2005, www.wallandbinkley.com/quaedam/?p=27 (accessed Nov. 11, 2005).

9. Francis Shanahan, Zuggest, www.francisshanahan.com/zuggest.aspx (accessed Apr. 5, 2006).

10. A. Rhyno, "Ajax and the Rich Web Interface," *LibraryCog blog* Apr. 10, 2005, http://librarycog.uwindsor.ca:8087/artblog/librarycog/1113186562 (accessed Nov. 11, 2005); R. Tennant, "Tennant's Top Tech Trend Tidbit," *LITA Blog* June 22, 2005, http://litablog.org/?p=35 (accessed Nov. 11, 2005).

11. T. Hickey, "Ajax and Web Interfaces," Outgoing blog, Mar. 31, 2005. Retrieved Nov. 11, 2005 http://outgoing.typepad.com/outgoing/2005/03/web_application.html.

12. OCLC DeweyBrowser. http://ddcresearch.oclc.org/ebooks/fileServer (accessed Apr. 5, 2006).

13. Hickey, "Ajax and Web Interfaces."

14. J. Wusteman, "From *Ghostbusters* to Libraries: The Power of XUL," *Library Hi Tech* 23, no 1 (2005a). Retrieved Nov. 11, 2005 www.ucd.ie/wusteman/; Cover Pages, Microsoft Extensible Application Markup Language (XAML), http://xml.coverpages.org/ms-xaml.html (accessed Apr. 5, 2006).

15. Google Extensions for Firefox, http://toolbar.google.com/firefox/extensions/index.html (accessed Apr. 5, 2006).

16. C. Adams, "Ajax: Usable Interactivity with Remote Scripting," *SitePoint.* (Jul. 13, 2005), www.sitepoint.com/article/remote-scripting-ajax (accessed Nov. 11, 2005).

17. XHTML 2.0, W3C Working Draft, May 27, 2005, www.w3.org/TR/2005/WD-xhtml2-20050527 (accessed Apr. 5, 2006).

18. Client/server model, http://en.wikipedia.org/wiki/Client/server (accessed Apr. 5, 2006).

19. DLL Hell, http://en.wikipedia.org/wiki/DLL_hell (accessed Apr. 5, 2006).

20. J. Nielsen, "The Need for Speed."

21. Service-Oriented Architecture, http://en.wikipedia.org/wiki/Service-oriented_architecture (accessed Apr. 5, 2006).

22. J. Wusteman, "Realizing the Potential of Web Services," *OCLC Systems & Services: International Digital Library Perspectives* 22, no. 1 (2006): 5–9.

23. ARC—A Cross Archive Search Service, Old Dominion University Digital Library Research Group, http://arc.cs.odu.edu (accessed Apr. 5, 2006); NISO MetaSearch Initiative, www.niso.org/committees/MS_initiative.html (accessed Apr. 5, 2006); ARC download page, SourceForge, http://oaiarc.sourceforge.net (accessed Apr. 5, 2006).

24. Open Archives Initiative Protocol for Metadata Harvesting, www.openarchives.org/OAI/openarchivesprotocol.html (accessed Apr. 5, 2006).

25. OJAX download page, SourceForge, http://sourceforge.net/projects/ojax (accessed Apr. 5, 2006).

26. Apache Jakarta Project, http://jakarta.apache.org (accessed Apr. 5, 2006); Apache Jakarta Commons DBCP, http://jakarta.apache.org/commons/dbcp (accessed Apr. 5, 2006); Apache Jakarta Commons DbUtils, http://jakarta.apache.org/commons/dbutils (accessed Apr. 5, 2006).

27. Agile software development definition, Wikipedia, http://en.wikipedia.org/wiki/Agile_software_development (accessed Apr. 5, 2006).

28. Prototype JavaScript Framework, http://prototype.conio.net (accessed Apr. 5, 2006); script.aculo.us, http://script.aculo.us (accessed Apr. 5, 2006); Rico, http://openrico.org/rico/home.page (accessed Apr. 5, 2006).

29. Sabre, www.sabre.com (accessed Apr. 5, 2006).

30. NISO MetaSearch Initiative, www.niso.org/committees/MS_initiative.html (accessed Apr. 5, 2006).

31. Google Toolbar, http://toolbar.google.com (accessed Apr. 5, 2006).

32. Dublin Core Metadata Initiative, http://dublincore.org (accessed Apr. 5, 2006).

33. MySQL, www.mysql.com (accessed Apr. 5, 2006).

34. Google Help Center, Advanced Operators, www.google.com/help/operators.html (accessed Apr. 5, 2006).

35. Apache Lucene, http://lucene.apache.org (accessed Apr. 5, 2006).

36. J. Nielsen, "Search: Visible and Simple," *Alertbox* May 13, 2001, www.useit.com/alertbox/20010513.html (accessed Nov. 11, 2005).

37. Francis Shanahan, Zuggest.

38. J. R. Baker, "The Impact of Paging versus Scrolling on Reading Online Text Passages," *Usability News* 5, no. 1 (2003), http://psychology.wichita.edu/surl/usabilitynews/51/paging_scrolling.htm (accessed Nov. 11, 2005); J. Nielsen, "Search: Visible and Simple."

39. J. J. Garrett, "Ajax: A New Approach to Web Applications."

40. del.icio.us, http://del.icio.us (accessed Apr. 5, 2006); Furl, www.furl.net (accessed Apr. 5, 2006).

41. Google Sitemaps (BETA) Help, www.google.com/webmasters/sitemaps/docs/en/other.html (accessed Apr. 5, 2006).

42. Ruby on Rails, www.rubyonrails.org (accessed Apr. 5, 2006); Java 2 Platform, Enterprise Edition (J2EE), http://java.sun.com/j2ee (accessed Apr. 5, 2006); M. LaMonica, "Microsoft Gets Hip to AJAX," CNET News.com, June 27, 2005, http://news.com.com/Microsoft+gets+hip+to+AJAX/2100-1007_3-5765197.html (accessed Nov. 11, 2005).