

Considering Ajax, Part 2: Change your life with mashups

Pitfalls to avoid -- and great opportunities for the future

Level: Introductory

[Chris Laffra \(laffrac@us.ibm.com\)](mailto:laffrac@us.ibm.com), Performance Engineering Team Lead, IBM

23 May 2006

Continue your exploration of what Ajax developers need to keep in mind when they build applications, in this article by Chris Laffra. In addition to concrete advice and warnings, catch a vision of Ajax's future, where it powers user-directed mashups of content on personalized Web pages.

In the first installment of this series, I talked about the hype by which Ajax is currently surrounded. You also learned that reliable frameworks are still under construction, and that you should worry about navigation history, bookmarkability, feedback, persistence, concurrency, and security.

In this follow-up article, the focus is on document repaints, testing, publish and subscribe issues, performance, accessibility, support for older browsers, and stating your intentions. I'll also touch on some interesting opportunities that Ajax offers for developing Web sites inside Web sites.

Getting lost

It's a bad idea for an application, rather than the user, to move the mouse cursor. This is a well-known rule in user interface (UI) design and you should avoid such moves at all cost. A similar rule for Ajax applications is: Never scroll my window for me. If an asynchronous Ajax response comes back from a server and text is pasted in a DOM element that is located above the current document view, the browser will scroll the contents downwards. The result can be highly disorienting.

The problem with Ajax is that it tries to treat a document as something it is not. Google Maps does not have this problem -- it hardly needs a scrollbar. This is because it really behaves like an application. However, read a long thread in Gmail, and your risk of getting lost is very realistic, as scrollbars are involved, and any click on a banner rearranges the contents of the window and its scrollbars.

Ajax applications that treat a document as an applications should avoid updating the document if the end result is in front of the current viewport. Viewport management and conditional DOM manipulation is not trivial, and therefore Ajax applications should prepare themselves to simply *ignore* responses that take too long to return. It is better to add a **Retry** or **Refresh** button for such contingencies.

Recommendation: Avoid doing smart things like moving the scrollbars explicitly.

Testing

"Never stop testing, and your products never stop improving." --David Ogilvy

Ajax development is multi-platform development, requiring multi-platform testing. Subtle differences, incompatible event models, and irreconcilable DOM incompatibilities force Ajax developers into writing defensive code that deals with the differences.

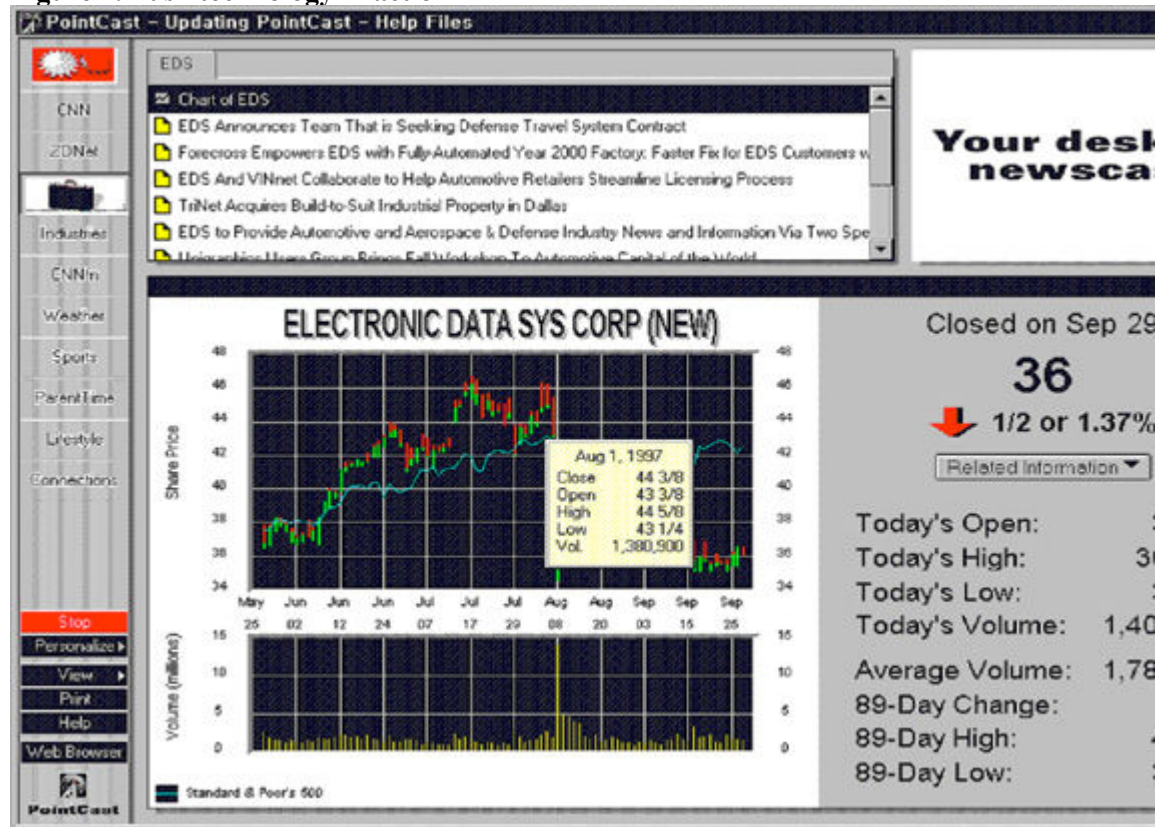
Most automated screen testing tools explicitly state that they do not support DHTML. With dynamic HTML that moves things around and dynamically changes the DOM structure, automated testing tools have no chance to make sense of what the UI looks like, and, more importantly, how they can drive it. Consider a site like Start.com and imagine how you would test it. I have no idea how you'd go about doing this, other than by hiring low-wage testing drones.

Though JavaScript makes automated testing more difficult, Ajax has an advantage: The componentization (atomization) of services on the server allows for finer-grained testing of the business logic. This allows for a higher degree of coverage and variability of functional testing.

Where's the push?

Figure 1 offers a blast from the past: push technology-based PointCast.

Figure 1. Push technology in action



PointCast created a fabulous stir around 1996 by offering a free reader that pushed free content (and advertisements) over the network. PointCast 2.6 required a PC with at least a 486/33 processor, 8 MB of RAM, and 10 MB of free disk space -- all incredibly small numbers by today's standards.

When you compare the PointCast UI with something like Start.com, the similarities are striking. It's valid to wonder what Ajax offers beyond PointCast. Except for higher fidelity in the form of high-resolution images, shading, and smooth dragging, Ajax essentially still offers the limited amount of information in which the user has indicated interest.

Ajax lacks support for publish-and-subscribe architectures, because other than HTTP streaming, it has no standard way to open a sustained socket connection to an IP address. Through Flash's `XMLSocket` object and

Flash/JavaScript Integration Kit, Ajax applications might open a long lasting full-duplex TCP/IP socket stream to an MQSeries server, for instance. However, an application built this way requires a Flash player to be installed in the client browser. Currently, all that Ajax applications can do is to poll the server from which they are loaded, essentially using the server as a proxy/gateway to a data server. This might put an excessive burden on the network and on servers that are unlikely to be able to withstand constant polling from thousands or (the price of success) millions of users. An alternative architecture is required.

CPU and leaks

Modern desktop machines are constantly getting faster, but moving code from server environments onto desktops means placing demands on an environment that is not as easily updated. Memory comes cheap -- around \$25 for 512 MB -- but installation in large installed bases is prohibitively expensive. Slower machines will have a hard time keeping up with the processing demands generated by all those nice DHTML animations and DOM manipulations. More code to send over, in addition to the content in HTML, will make the Web site slower to load.

Unbeknownst to most, JavaScript has a tendency to leak. If you're not careful, the browser's memory demands will grow over the duration of a browsing session. CPU performance limited the adoption of JavaScript in the past. As browsers become faster and faster, partly thanks to thriving competition, remember that many user desktops are not as well equipped as the typical developer machine.

Accessibility

Accessibility for people with disabilities is not a topic the average developer pays much attention to. But developers face moral, if not legal requirements as they consider what technical preparations to take so a Web site is accessible to disabled users. Due to its declarative nature, HTML lends itself very well to simple techniques that solve these problems: it can be processed by screen readers and displayed with large fonts for visually impaired users, for instance.

The addition of dynamic HTML to a Web site will continuously change its structure. Braille computers, screen readers, and navigation tools based on spoken instructions find this continuous change hard to deal with.

JavaScript: Not everywhere yet

Browser statistics from thecounter.com for October 2005 show that 10 percent of all Web browsers have JavaScript turned off or otherwise unavailable. This is better than the 15 percent reported for 2001, but Web sites still have to contend with users who do not want to run JavaScript.

One well-known subclass of browsers that lack JavaScript support is search robots. A Web site that relies on Ajax techniques will have to put in extra care to ensure that it still shows up in search queries. Keywords in meta headers can help direct search engines. On the other hand, Ajax might be useful for *hiding* certain content from prying search robots.

Intentions

Traditionally, links on the Web take you somewhere else, and all users know and expect this. Moreover, the underlying GET requests are supposed to be stateless. Although this is not covered under a standard as far as I know, browsers and search engines expect links, which are served by GET requests, to be stateless. If an Ajax application starts to use links to make state changes, the various parties involved will become confused, including the poor user.

Do not expect users to be flexible and to understand how to personalize their experience by dragging things on the screen. They might not even want to do any of this. Furthermore, you need to inform users in clear and obvious terms what to do, which is perhaps also a reason why so many road signs in the U.S. include written text. Metaphors from one domain -- say, Linux/PHP script hackers -- will not apply at all to another -- say, 50-year-old insurance brokers. Some visual metaphors can be cool in one culture, but downright offensive in others.

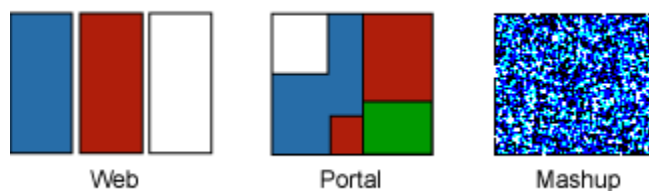
In short, you need to clearly state your intentions, keep interactions as trivial as possible, and assume that your users include novice and inexperienced users. Personalization is important, but do not add it to your application at the cost of usability, nor allow it to impede the implementation of something like support for internationalization.

Mashup

"Today, most software exists, not to solve a problem, but to interface with other software." -- I. O. Angell

The subliminal message that most Ajax applications send out is the power of *aggregation*. Often, advantages of Ajax are described in terms of reducing round trip costs to the server. However, the real benefits will come when Ajax applications move beyond the stage of being simple portals to performing truly transparent client-side *mashup*, as illustrated in [Figure 2](#).

Figure 2. Ajax's full potential?



For example, personalized Google and Start.com each perform a Web search and then save the result in the current page. Cookies and server-side storage or Flash objects can persist the current page layout, so that the next time you visit the page, your preferred mashup is maintained. Both Google and Start.com offer user-defined gadgets that allow developers to define their own portlets. Still, this does not bring you beyond the stage of a portal.

For Ajax to become truly successful, it needs to disappear. That needs client-side, user-programmable versions of Greasemonkey (see [Resources](#)) that allow designers and power users to aggregate their own information sources and to define their own presentation. The functions are coming closer, but simpler programming models are needed before people other than the geeks at Google or Microsoft can develop these apps.

As a concrete example, I set up a [client-driven mashup demo](#) for you to look at. Rather than rely on a server to

create a custom page, the page composition is done entirely in JavaScript. This sample illustrates how to collect information from (a) HTML, as it uses weather.com to get localized temperatures; (b) SQL databases running on a remote server; and (c) RSS feeds -- in this case, using those feeds to aggregate poker news. What I tried to do is unify information collection, transformation, and mashing up in a unified fashion. Check out the source code available on the demo page to see how declarative templates do HTML conversion and some simple business logic (click the `src` links on the demo page).

In situ editing

Many wikis allow users to edit the contents of a page fragment in rich text editors and publish the result on the server, all without leaving the browser. Similarly, you could compose applications (if you want to call them that) without ever installing a development environment. Ajax techniques allow for more and more aspects of Web site design to be defined in the same environment in which they are displayed. I call this concept *in situ editing*. Others refer to the phenomenon as *situational programming*. I provided a [quick demo in which you can edit a DOM fragment](#) so you can see how it works.

Persistence

A great example of how HTML documents can self-modify and persist themselves on the filesystem is TiddlyWiki (see [Resources](#)). If you view TiddlyWiki, you've already downloaded the entire application, as it works entirely as a JavaScript application inside a Web page. The HTML is edited in situ (that is, in the same browser window where you viewed it only a few seconds earlier), and is saved on the local filesystem.

If you prefer not to save material on your local hard disk, you can use techniques such as WebDAV to share and version resources on the Web. Using WebDAV is but one of the available techniques for uploading a new version of a Web page after it is updated through an in situ editing session. An alternative is to upload the page yourself using `XmlHttpRequest` and connect to a REST service. The latter technique requires a specialized server to receive your XML requests and maintain different versions of pages, though you can easily write such a server extension in a few lines of PHP.

I've provided [an example](#) that illustrates how to extract a document's DOM from the current page, transmit it to a server, and to reload it later, even in another browser. Check out the included sources on the sample page to see how easily you can implement this.

In conclusion

By now, you should have learned all the potential problem areas to consider when you apply Ajax techniques to your latest Web site project. Don't be too discouraged; Ajax encompasses a set of wonderful techniques to make a Web site more responsive and useful, and to explore rich client application development inside a browser. Follow the example of successful Ajax applications, adopt the things that work, and be careful not to repeat the mistakes of others. If you can do all that, you will create the coolest Web site your grandma ever saw!

Resources

Learn

- The author wrote three Ajax demo applications to help you understand the concepts in this article and see Ajax's potential:
 - [A client-driven mashup demo](#)
 - [A quick demo in which you can edit a DOM fragment](#)
 - [An example that illustrates how to extract a document's DOM from the current page, transmit it to a server, and reload it later, even in another browser](#)
- ["Considering Ajax, Part 1: Cut through the hype"](#) (Chris Laffra, developerWorks, May 2006): Read the first article in this series for more on when and how to implement this new technology.
- [Alex Bosworth's list of Ajax mistakes](#): Explore part of the inspiration for this series.
- [Web Accessibility Initiative](#): Get information on how to make Web sites accessible to the disabled.
- [mashup](#): Read a good explanation of what this means in a Web application context on Wikipedia.
- [thecounter.com](#): Check out browser stats.
- developerWorks [Web Architecture zone](#): Expand your site development skills with articles and tutorials that specialize in Web technologies.
- [developerWorks technical events and webcasts](#): Stay current with jam-packed technical sessions that shorten your learning curve, and improve the quality and results of your most difficult software projects.

Get products and technologies

- [Google Maps](#): Check out a successful and influential Ajax application.
- [Greasemonkey](#): Try to analyze Ajax applications running online with this useful tool.
- [TiddlyWiki](#): Experiment and create personal self-contained hypertext documents.

Discuss

- [Participate in the discussion forum](#).
- [developerWorks blogs](#): Get involved in the developerWorks community.
- [developerWorks discussion forums](#): Join the discussion threads that interest you.

About the author

Chris Laffra was born in the Netherlands and obtained his MSc at the Vrije Universiteit of Amsterdam in 1988 and a PhD at the Erasmus University of Rotterdam in 1992. At both IBM T.J. Watson Research Center and Morgan Stanley, Chris worked on tools for user interfaces, component infrastructures, program analysis, debugging, visualization, compression, and optimization. He led the OTI Amsterdam lab for three and a half years, working on WebSphere Studio Device Developer. At IBM Canada's lab in Ottawa, he worked on the border between Java runtime environments and Eclipse (and co-authored the Official Eclipse 3.0 FAQs). Currently, Chris works as Performance Engineering Team Lead at IBM Rational to improve RAD/RSA

performance. He has experimented a lot with Ajax in the past, doing things such as enhancing Google Maps to find a new home in Raleigh near a good school, and generating the [online version of the Eclipse FAQs](#).
