Ajax Mistakes

Mistakes made when developing with Ajax .

Mistakes made when developing with Ajax .

Mistakes made when developing with Ajax .

Using Ajax for the sake of Ajax.

Sure Ajax is cool, and developers love to play with cool technology, but Ajax is a tool, not a toy. A lot of Ajax isn't seriously needed to improve usability but rather experiments in what Ajax can do or trying to fit Ajax somewhere where it isn't needed.

Breaking the back button

The back button is a great feature of the standard web site user interface. Unfortunately, the back button doesn't mesh very well with Javascript. Keeping back button functionality is one reason not to go with a pure Javascript web app.
Keep in mind however that good web design provides the user with everything they would need to successfully navigate your site, and never relies on web browser controls.

Not giving immediate visual cues for clicking widgets

If something I'm clicking on is triggering Ajax actions, you have to give me a visual cue that something is going on. An example of this is GMail loading button that is in the top right. Whenever I do something in GMail, a little red box in the top right indicates that the page is loading, to make up for the fact that Ajax doesn't trigger the normal web UI for new page loading.

Leaving **offline** people behind

As web applications push the boundaries further and further, it becomes more and more compelling to move all applications to the web. The provisioning is better, the world-wide access model is great, the maintenance and configuration is really cool, the user interface learning curve is short.

However, with this new breed of Ajax applications, people who have spotty internet connections or people who just don't want to switch to the web need to be accomodated as well. Just because technology 'advances' doesn't mean that people are ready and willing to go with it. Web application design should at least consider offline access. With GMail it's POP, Backpackit has SMS integration. In the Enterprise, it's web-services.

Don't make me wait

With Firefox tabs, I can manage various waits at websites, and typically I only have to wait for a page navigation. With AJAX apps combined with poor network connectivity/bandwidth/latency I can have a really terrible time managing an interface, because every time I do something I have to wait for the server to return a response. However, remember that the 'A' in AJAX stands for 'Asynchronous', and the interaction can be designed so that the user is not prevented from continuing to work on the page while the earlier request is processed.

Sending sensitive information in the clear

The security of AJAX applications is subject to the same rules as any web application, except that once you can talk asynchronously to the server, you may tend to write code that is very chatty in a potentially insecure way. All traffic must be vetted to make sure security is not compromised.

Assuming AJAX development is single platform development.

Ajax development is multi-platform development. Ajax code will run on IE's javascript engine, Spidermonkey (Mozilla's js engine), Rhino (a Java js implementation, also from Mozilla), or other minor engines that may grow into major engines. So it's not enough just to code to JavaScript standards, there needs to be real-world thorough testing as well. A major obstacle in any serious Javascript development is IE's buggy JS implementation, although there are tools to help with IE JS development.

Forgetting that multiple people might be using the same application at the same time

In the case of developing an Intranet type web application, you have to remember that you might have more than one person using the application at once. If the data that is being displayed is dynamically stored in a database, make sure it doesn't go "stale" on you.

Too much code makes the browser slow

Ajax introduces a way to make much more interesting javascript applications, unfortunately interesting often means more code running. More code running means more work for the browser, which means that for some javascript intensive websites, especially inefficiently coded ones, you need to have a powerful CPU to keep the functionality zippy. The CPU problem has actually been a limit on javascript functionality in the past, and just because computers have gotten faster doesn't mean the problem has disappeared.

Not having a plan for those who do not enable or have JavaScript.

According to the W3 schools browser usage statistics, which if anything are skewed towards advanced browsers, 11% of all visitors don't have JavaScript. So if your web application is wholly dependent on JavaScript, it would seem that you have potentially cut a tenth of your audience.

Blinking and changing parts of the page unexpectedly

The first A in Ajax stands for asynchronous. The problem with asynchronous messages is that they can be quite confusing when they pop in unexpectedly. Asynchronous page changes should only ever occur in narrowly defined places and should be used judiciously, flashing and blinking in messages in areas I don't want to concentrate on harkens back to days of the html blink tag. "Yellow Fade", "One Second Spotlight" and other similar techniques are used to indicate page changes unobtrusively.

Not using links I can pass to friends or bookmark

Another great feature of websites is that I can pass URLs to other people and they can see the same thing that I'm seeing. I can also bookmark an index into my site navigation and come back to it later. Javascript, and thus Ajax applications, can cause huge problems for this model of use. Since the Javascript is dynamically generating the page instead of the server, the URL is cut out of the loop and can no longer be used as an index into navigation. This is a very unfortunate feature to lose, many Ajax webapps thoughtfully include specially constructed permalinks for this exact reason.

Blocking Spidering

Ajax applications that load large amounts of text without a reload can cause a big problem for search engines. This goes back to the URL problem. If users can come in through search

engines, the text of the application needs to be somewhat static so that the spiders can read it.

Asynchronously performing batch operations

Sure with Ajax you can make edits to a lot of form fields happen immediately, but that can cause a lot of problems. For example if I check off a lot of check boxes that are each sent asynchronously to the server, I lose my ability to keep track of the overall state of checkbox changes and the flood of checkbox change indications will be annoying and disconcerting.

Scrolling the page and making me lose my place

Another problem with popping text into a running page is that it can effect the page scroll. I may be happily reading an article or paging through a long list, and an asynchronous javascript request will decide to cut out a paragraph way above where I'm reading, cutting my reading flow off. This is obviously annoying and it wastes my time trying to figure out my place. But then again, that would be a very stupid way to program a page, with or without AJAX.

Inventing new UI conventions

A major mistake that is easy to make with Ajax is: 'click on this non obvious thing to drive this other non obvious result'. Sure, users who use an application for a while may learn that if you click and hold down the mouse on this div that you can then drag it and permanently move it to this other place, but since that's it's not in the common user experience, you increase the time and difficulty of learning the application, which is a major negative for any application. On the plus side, intuitiveness is a function of learning, and AJAX is popularising many new conventions which will become intuitive as time goes by. The net result will be greater productivity once the industry gets over the intuitiveness hump.

Character Sets

One big problem with using AJAX is the lack of support for character sets. You should always set the content character set on the server-side as well as encoding any data sent by Javascript. Use ISO-8859-1 if you use plain english, or UTF-8 if you use special characters, like æ, ø and å (danish special characters) Note: it is usually a good idea to go with utf-8 nowadays as it supports many languages).

Changing state with links (GET requests)

The majority of Ajax applications tend to just use the GET method when working with AJAX. However, the W3C standards state that GET should only be used for retrieving data, and

POST should only be used for setting data. Although there might be no noticable difference to the end user, these standards should still be followed to avoid problems with robots or programs such as Google Web Accelerator.

Not cascading local changes to other parts of the page

Since Ajax/Javascript gives you such specific control over page content, it's easy to get too focused on a single area of content and miss the overall integrated picture. An example of this is the Backpackit title. If you change a Backpackit page title, they immediately replace the title, they even remember to replace the title on the right, but they don't replace the head title tag with the new page title. With Ajax you have to think about the whole picture even with localized changes.

Problem reporting

In a traditional server-side application, you have visibility into every exception, you can log all interesting events and benchmarks, and you can even record and view (if you wish) the actual HTML that the browser is rendering. With client-side applications, you may have no idea that something has gone wrong if you don't know how to code correctly and log exceptions from the remotely called pages to your database.

Return on Investment

Sometimes AJAX can impressively improve the usability of an application (a great example is the star-rating feedback on Netflix), but more often you see examples of expensive rich-client applications that were no better than the plain HTML versions.

Mimicing browser page navigation behavior imperfectly

One example of this is blinklist Ajax paging mechanism on the front page. As you click to see another page of links, ajax fills in the next page. Except that if you are used to a browser experience, you probably expect to go to the top of the page when you hit next page, something JavaScript driven page navigation doesn't do. BlinkList actually anticipates this and tries to counteract by manipulating your scrolling to scroll upwards until you hit the top. Except this can be slow and if you try scrolling down you will fight the upwards scrolling JavaScript and it won't let you scroll down. But then again, that is very stupid way to program a page, with or without AJAX.

Another Tool

It seems everyone has forgotten that Ajax is just another tool in the toolbox for Web Design. You can use it or not and misuse it or not. The old 80/20 rule always applies to applications (if you cover 80% of what all users want/need then you have a viable app) and if you lose 11% of your audience because they don't switch on their javascript then you have to ask yourself if changing your app is worth capturing that 11% or stick with 89% that are currently using it and move on to something else. Also web apps should take advantage of all tricks to enable them to function quickly and efficiently. If that means using javascript for some part, Ajax for another and ASP callbacks for a third, so be it.