AJAX Programming Overview

Introduction

Overview

In the world of Web programming, AJAX stands for Asynchronous JavaScript and XML, which is a technique for developing more efficient interactive Web applications. AJAX enables complex interactive Web site elements to remain loaded while switching between pages, so that they do not have to be served up separately each time a visitor navigates to another site page. This Personal Learning Resource gives a brief overview of AJAX, discusses its advantages and disadvantages, and also lists numerous other resources for additional training on this development method.

AJAX (Asynchronous JavaScript And XML) Key Components

AJAX itself is not considered to be a unique technology, but a Web development method incorporating features from several different technologies and languages. AJAX uses a communication technology (typically SOAP and XML) to send and receive an asynchronous request/response to the server, and then leverages presentation technologies (JavaScript, DOM, HTML, and CSS) to process the response. The AJAX method implements the following technologies to ease the process of producing consistent and interactive Web pages:

1. XHTML (HTML) and CSS, for marking up and styling information.
2. The DOM accessed with a client-side scripting language, especially ECMAScript implementations like JavaScript andJScript, to dynamically display and interact with the information presented.
3. The XMLHttpRequest object to exchange data asynchronously with the Web server. In some AJAX frameworks and in certain situations, an iFrame object is used instead of the XMLHttpRequest object to exchange data with the Web server.
4. XML is commonly used as the format for transferring data back from the server, although any format will work, including preformatted HTML, plain text, JSON and even EBML.

Using Ajax technologies in Web applications provides many challenges for developers interested in adhering to WAI accessibilityguidelines. Developers need to provide fallback options for users on other platforms or browsers, as most methods of AJAX implementation rely on features only present in desktop graphical browsers.

Advantages

- **Portability** — AJAX applications use well-documented features present in all major browsers on most existing platforms. Though this situation could feasibly change in the future, at the moment, AJAX applications are effectively cross-platform. While the Ajax platform is more restricted than the Java platform, current AJAX applications effectively fill part of the one-time niche of Java applets: extending the browser with lightweight mini-applications.
- **Interactivity** — Ajax applications are mainly executed on the user's machine, by manipulating the current page within their browser using document object model methods. AJAX can be used for a multitude of tasks such as updating or deleting records; expanding Web forms; returning simple search queries; or editing category trees — all without the requirement to fetch a full page of HTML each time a change is made. Generally only small requests are required to be

sent to the server, and relatively short responses are sent back. This permits the development of more interactive applications featuring more responsive user interfaces due to the use of DHTML techniques.

Concerns/Issues

- **Browser Usability** — One major complaint voiced against the use of AJAX in Web applications is that it might easily break the expected behavior of the browser's back button. The different expectations between returning to a page which has been modified dynamically versus the return to a previous static page might be a subtle one. Users generally expect that clicking the back button in Web applications will undo their last state change and in AJAX applications this might not be the case. Developers have implemented various solutions to this problem, most of which revolve around creating or using invisible iFRAMEs to invoke changes that populate the history used by a browser's back button.

  A related issue is that dynamic Web page updates make it difficult for a user to bookmark a particular state of the application. Solutions to this problem exist, many of which use the URL fragment identifier to keep track of, and allow users to return to, the application in a given state. This is possible because many browsers allow JavaScript to update the fragment identifier of the URL dynamically, so that AJAX applications can maintain it as the user changes the application's state. This solution also improves back-button support.

- **Response time** — Network latency — or the interval between user request and server response — needs to be considered carefully during AJAX development. Without clear feedback to the user, smart preloading of data, and proper handling of the XMLHttpRequest object, users might experience delay in the interface of the Web application. The use of visual feedback to alert the user of background activity and/or preloading of content and data are often suggested solutions to these latency issues.
- **JavaScript** — While no browser plug-in is required for AJAX, it requires users to have JavaScript enabled in their browsers. This applies to all browsers that support AJAX except for Microsoft Internet Explorer 6 and below which additionally require ActiveX to be enabled, as the XMLHttpRequest object is implemented with ActiveX in this browser.

  As with DHTML applications, AJAX applications must be tested rigorously to deal with the quirks of different browsers and platforms. A number of programming libraries have become available as AJAX has matured that can help ease this task. Likewise, techniques have been developed to assist in designing applications which degrade gracefully and offer alternative functionality for users without JavaScript enabled.

Web developers use AJAX in some instances to provide content only to specific portions of a Web page, allowing data manipulation without incurring the cost of re-rendering the entire page in the Web browser. Non-AJAX users would optimally continue to load and manipulate the whole page as a fallback, allowing the developers to preserve the experience of users in non-AJAX environments (including all relevant accessibility concerns) while giving those with capable browsers a much more responsive experience.
If anything about current interaction design can be called “glamorous,” it’s creating Web applications. After all, when was the last time you heard someone rave about the interaction design of a product that wasn’t on the Web? (Okay, besides the iPod.) All the cool, innovative new projects are online.

Despite this, Web interaction designers can't help but feel a little envious of our colleagues who create desktop software. Desktop applications have a richness and responsiveness that has seemed out of reach on the Web. The same simplicity that enabled the Web's rapid proliferation also creates a gap between the experiences we can provide and the experiences users can get from a desktop application.

That gap is closing. Take a look at Google Suggest. Watch the way the suggested terms update as you type, almost instantly. Now look at Google Maps. Zoom in. Use your cursor to grab the map and scroll around a bit. Again, everything happens almost instantly, with no waiting for pages to reload.

Google Suggest and Google Maps are two examples of a new approach to web applications that we at Adaptive Path have been calling Ajax. The name is shorthand for Asynchronous JavaScript + XML, and it represents a fundamental shift in what's possible on the Web.

Defining Ajax

Ajax isn't a technology. It's really several technologies, each flourishing in its own right, coming together in powerful new ways. Ajax incorporates:

standards-based presentation using XHTML and CSS;

dynamic display and interaction using the Document Object Model;

data interchange and manipulation using XML and XSLT;

asynchronous data retrieval using XMLHttpRequest;

and JavaScript binding everything together.

The classic web application model works like this: Most user actions in the interface trigger an HTTP request back to a web server. The server does some processing—retrieving data, crunching numbers, talking to various legacy systems—and then returns an HTML page to the client. It's a model adapted from the Web's original use as a hypertext medium, but as fans of The Elements of User Experience know, what makes the Web good for hypertext doesn't necessarily make it good for software applications.

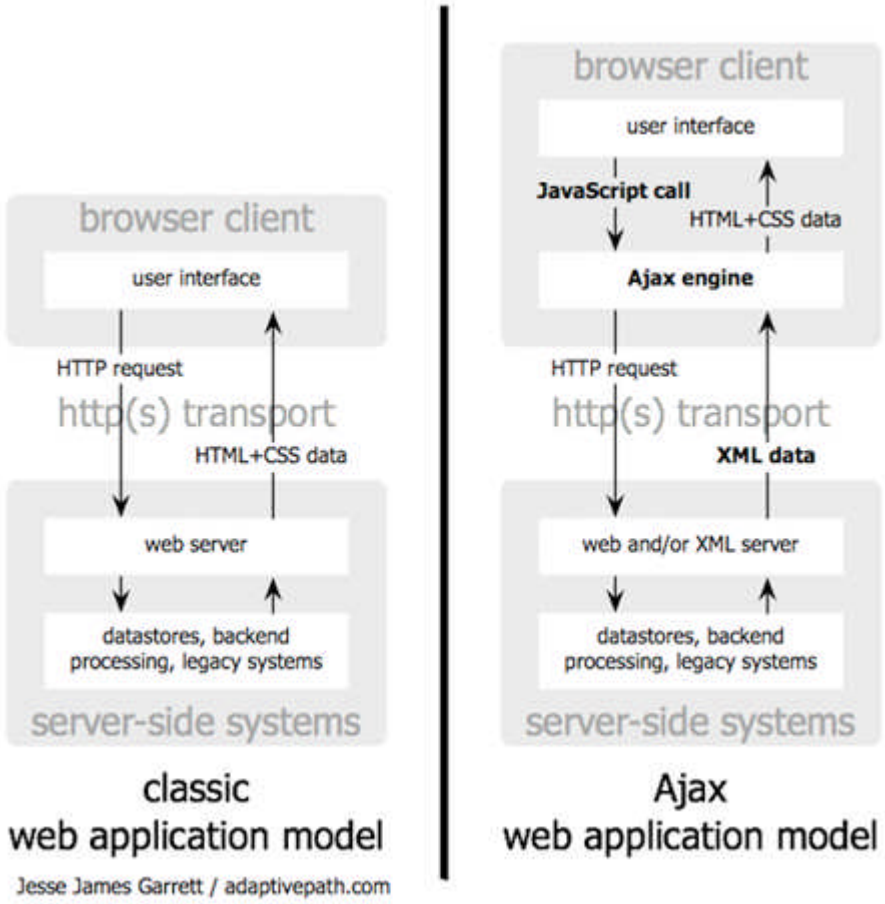

Figure 1: The traditional model for web applications (left) compared to the Ajax model (right).

This approach makes a lot of technical sense, but it doesn't make for a great user experience. While the server is doing its thing, what's the user doing? That's right, waiting. And at every step in a task, the user waits some more.

Obviously, if we were designing the Web from scratch for applications, we wouldn't make users wait around. Once an interface is loaded, why should the user interaction come to a halt every time the application needs something from the server? In fact, why should the user see the application go to the server at all?

How Ajax is Different

An Ajax application eliminates the start-stop-start-stop nature of interaction on the Web by introducing an intermediary—an Ajax engine—between the user and the server. It seems like adding a layer to the application would make it less responsive, but the opposite is true.

Instead of loading a webpage, at the start of the session, the browser loads an Ajax engine—written in JavaScript and usually tucked away in a hidden frame. This engine is responsible for both rendering the interface the user sees and communicating with the server on the user's behalf. The Ajax engine allows the user's interaction with the application to happen asynchronously—independent of communication with the server. So the user is never staring at a blank browser window and an hourglass icon, waiting around for the server to do something.

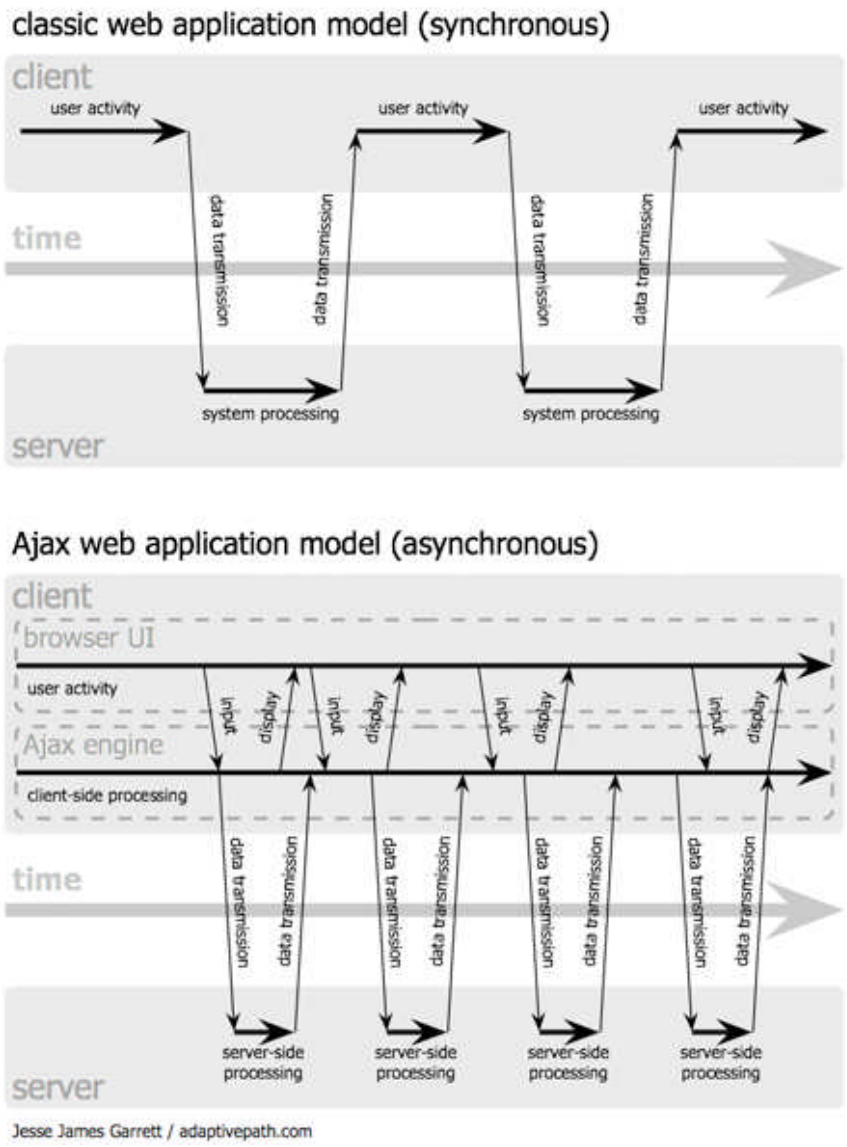

Figure 2: The synchronous interaction pattern of a traditional web application (top) compared with the asynchronous pattern of an Ajax application (bottom).

Every user action that normally would generate an HTTP request takes the form of a JavaScript call to the Ajax engine instead. Any response to a user action that doesn't require a trip back to the server—such as simple data validation, editing data in memory, and even some navigation—the engine handles on its own. If the engine needs something from the server in order to respond—if it's submitting data for processing, loading additional interface code, or retrieving new data—the engine makes those requests asynchronously, usually using XML, without stalling a user's interaction with the application.

Who's Using Ajax

Google is making a huge investment in developing the Ajax approach. All of the major products Google has introduced over the last year—Orkut, Gmail, the latest beta version of Google Groups, Google Suggest, and Google Maps—are Ajax applications. (For more on the technical nuts and bolts of these Ajax implementations, check out these excellent analyses of Gmail, Google Suggest, and

Google Maps.) Others are following suit: many of the features that people love in Flickr depend on Ajax, and Amazon's A9.com search engine applies similar techniques.

These projects demonstrate that Ajax is not only technically sound, but also practical for real-world applications. This isn't another technology that only works in a laboratory. And Ajax applications can be any size, from the very simple, single-function Google Suggest to the very complex and sophisticated Google Maps.

At Adaptive Path, we've been doing our own work with Ajax over the last several months, and we're realizing we've only scratched the surface of the rich interaction and responsiveness that Ajax applications can provide. Ajax is an important development for Web applications, and its importance is only going to grow. And because there are so many developers out there who already know how to use these technologies, we expect to see many more organizations following Google's lead in reaping the competitive advantage Ajax provides.

Moving Forward

The biggest challenges in creating Ajax applications are not technical. The core Ajax technologies are mature, stable, and well understood. Instead, the challenges are for the designers of these applications: to forget what we think we know about the limitations of the Web, and begin to imagine a wider, richer range of possibilities.