Ajax: A New Approach to Web Applications

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.

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?

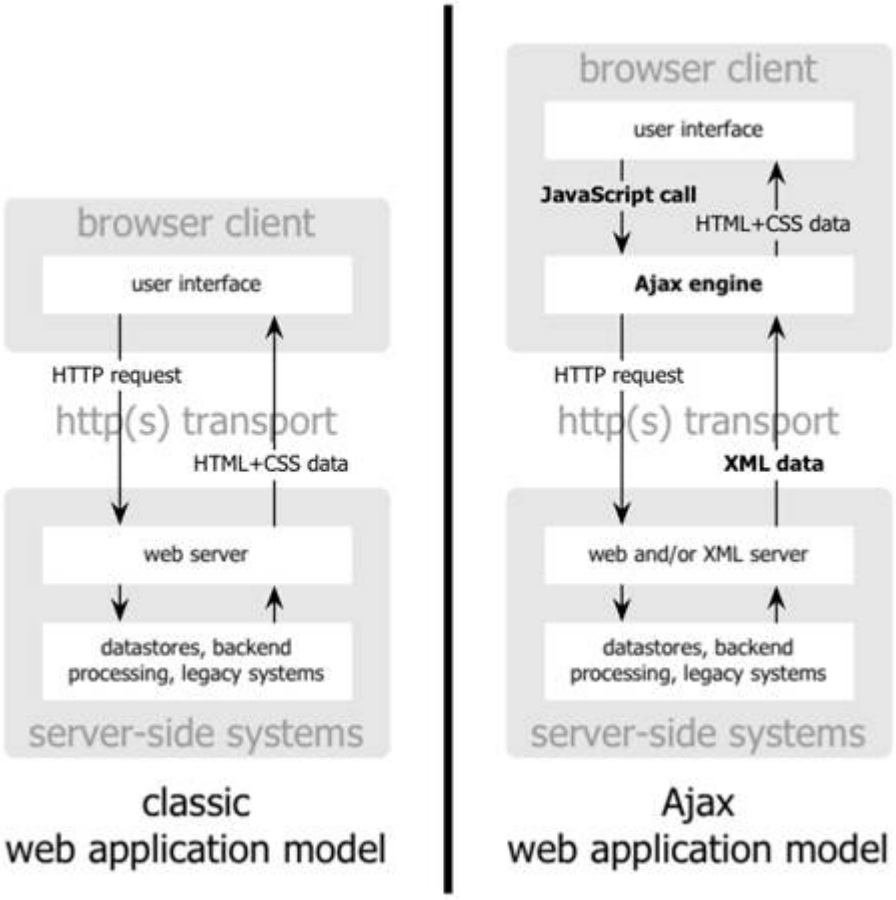

Figure 1: The traditional model for web applications (left) compared to the Ajax model (right).

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.

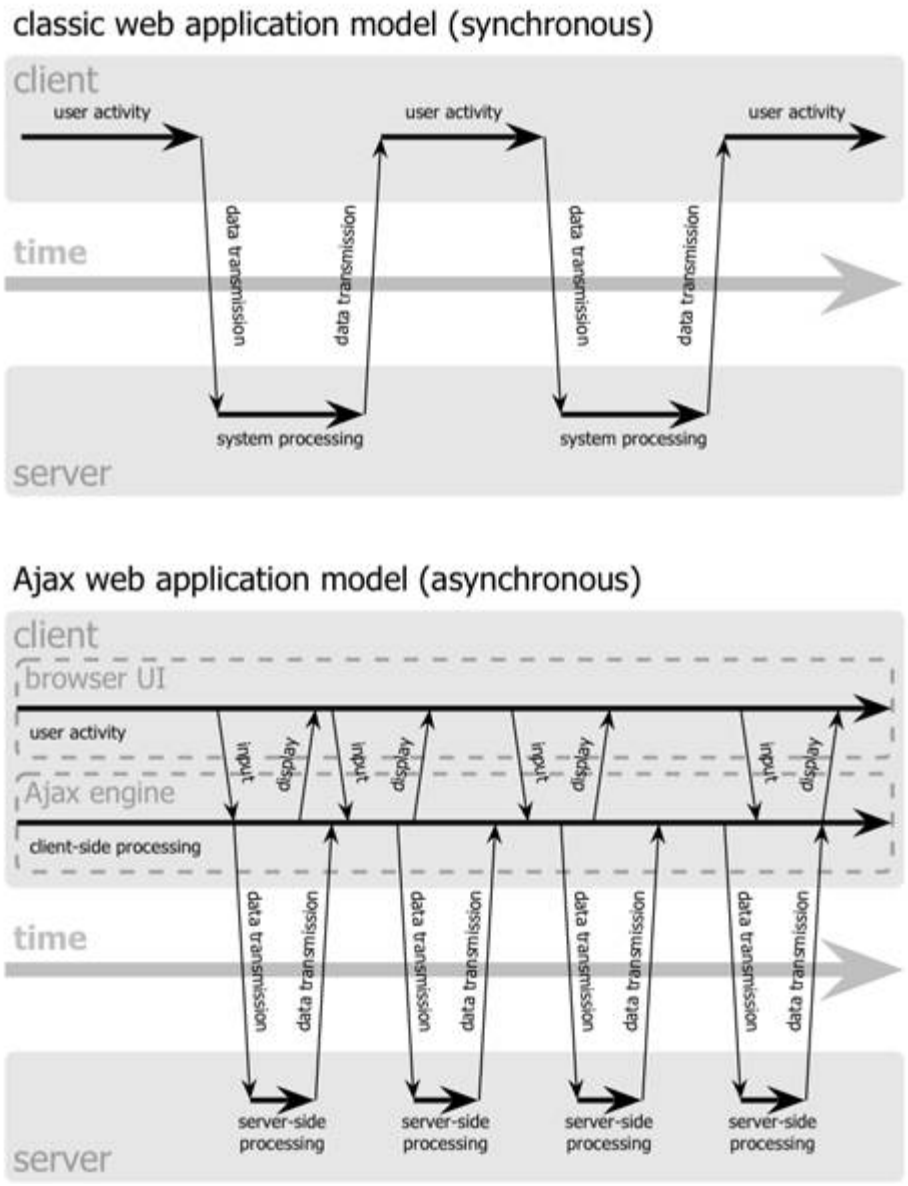

Figure 2: The synchronous interaction pattern of a traditional web application (top) compared with the asynchronous pattern of an Ajax application (bottom).

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.

Reference: http://www.adaptivepath.com/publications/essays/archives/000385.php

Thanks,
Shelly