Ajax at Work: Healthcare Provider's Dashboard

Doug Forst
Computing and Information Systems
University of Wisconsin Stevens Point
Stevens Point, WI 54481
Douglas.J.Forst@uwsp.edu

Josh Eide
Computing and Information Systems
University of Wisconsin Stevens Point
Stevens Point, WI 54481
joshua.t.eide@uwsp.edu

Robert Dollinger
Computing and Information Systems
University of Wisconsin Stevens Point
Stevens Point, WI 54481
rdolling@uwsp.edu

Abstract

AJAX is a set of technologies that combine, and work together, to leverage browser capabilities in order to reduce the amount and frequency of server postbacks that entirely recreate the page each time. The **Healthcare Provider's Dashboard** is a highly interactive custom application that uses Web Parts to display and manage a rich content of information associated with a healthcare provider. The application exposes an interface that consists of several parts, each with its specific information content that has a dynamic of its own. In this paper we present our experience of converting the Healthcare Provider's Dashboard into an AJAX enabled application. At this time, there are several approaches available to a developer who wants to apply AJAX to a web application. Two alternatives have been considered for our project: AJAX.NET and Microsoft's ASP.NET AJAX (former ATLAS). The major trade-offs and benefits of the two approaches are examined and evaluated.

1 Introduction

Web applications become richer and richer in the information content they provide. This requires new and innovative techniques to manage the increased amount and variety of information. The response, technology now provides, to the challenge of Rich Internet Applications (RIA) requirements, is a new web development methodology called AJAX (Asynchronous JavaScript And XML). AJAX is a set of technologies that combine, and work together, to leverage browser capabilities in order to reduce the amount and frequency of server postbacks that entirely recreate the page each time. With AJAX, regular postback cycles are eliminated or replaced by requests, typically asynchronous, for some specific data that result in partial updates on the page. This considerably reduces the load on the Web Server and enhances scalability of the server side resources, while making interactive Web applications more responsive.

The **Healthcare Provider's Dashboard** is a highly interactive custom application that uses Web Parts to display and manage a rich content of information associated with a healthcare provider. The application exposes an interface that consists of several parts, each with its specific information content that has a dynamic of its own. Parts can be dynamically removed from and recalled to the page at any time, and the information content of any part or RSS feed can be changed independently of the rest of page. In a traditional approach any of the above actions requires a server postback involving a complete rebuild of the page even when only one of the Web Parts is changing. With AJAX one can have the browser emit an XmlHttpRequest object which triggers the Web Server to retrieve only the specific information that needs to be changed on the page. This is remitted back to the browser in a mutually agreed upon format (XML, JSON or any kind of delimited string) where it is properly processed by a JavaScript for display. This process is much more efficient than rendering the entire Web page. In addition the XmlHttpRequest is typically asynchronous, which allows the user to continue his work on the page while some part of it is seamlessly changed behind the scenes. This results in a considerably enhanced user experience.

In this paper we present our experience of converting the Healthcare Provider's Dashboard into an AJAX enabled application. At this time, there are several approaches available to a developer who wants to apply AJAX to a web application. Two alternatives have been considered for our project: AJAX.NET and Microsoft's ASP.NET AJAX (former ATLAS).

The major trade-offs and benefits of the two approaches are examined and evaluated

The problem with the basic model of the Web, i.e. HTML and the request/response mechanism is that it has not been meant for and developed for what it is currently very often used. A Web site is nothing else but a bunch of pages linked together with the purpose of communicating some information. This is what the Web has been developed for initially. A Web application is an application developed by using Web technologies. However, applications need to conform to some well established usability criteria and

provide a certain level of interaction with the user. This where AJAX technologies can make a difference.
AJAX enables developers to build rich Web applications that have many advantages over the traditional Web applications that are completely server-based. Obviously, if we were designing the Web from scratch for applications, we wouldn't make users wait. 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? The idea that we are trying to pursue with Ajax is a dashboard that will have no lag time, which optimizes the users experience.

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.

2 Ajax

Ajax, shorthand for Asynchronous JavaScript and XML, is a web development technique for creating interactive web applications. The intent is to make web pages feel more responsive by exchanging small amounts of data with the server behind the scenes, so that the entire web page does not have to be reloaded each time the user requests a change. This is meant to increase the web page's interactivity, speed, and usability [7]. It has many advantages, few disadvantages, and the concept is fairly simple.

2.1 Advantages

Ajax is asynchronous, has minimal data transfer, limits processing on the server, increases responsiveness, and maintains the context of the current application. First Ajax uses asynchronous calls to the web server. This allows the user to continue viewing and also allows other actions to be done on the page before the first data has been completely updated. Also through asynchronous call backs the page seems very responsive. Next the minimal data transfer is achieved through Ajax. Because we are not performing a full postback and resending all the data to the server, the network operation is optimized and quicker operations is achieved. This can help with many places that allow minimal data transfer.

With the minimal data transfer we achieve limited processing on the server. Only the necessary data being sent back to the server, it is not required to process all elements. This in return minimizes the amount of processing on the server. The many items that would need to normally be sent back like images, elements, and also re-process the rest of the page is not needed because of the partial postback. Finally the context for the user is not changed when sending data back to the server. Normally when clicking submit when viewing a page the user is sent to a refreshed page, as if they were viewing it for the first time. In an Ajax application the user is able to maintain its current position. Therefore the user state is maintained, which optimizes the users experience [8].

2.2 Disadvantages

Ajax though has many advantages there are a couple disadvantages. Ajax does not support mobile systems. In order to run Ajax a user needs to have the transport mechanism and JavaScript support. Many mobile browsers do not have support for either. Because Ajax relies on JavaScript, which is implemented differently in different browsers many times there is the need to write multiple JavaScript methods. Also this causes the application to be tried and tested in the many different browsers. Another issue is the response time lag. Normally, a browser refreshes the whole page and the user waits to work on the new page. We with Ajax the user does not see the page refresh and therefore does not understand why they can not continue to work. Atlas solves this with the update progress toolkit. Overall there are just minor issues that are seen with Ajax. The two main ones that are explained here are solved by Atlas [7].

2.3 Framework

There are four main technological elements or ingredients that make up Ajax. Typically AJAX is referred to as an umbrella term for any combination of these technologies. The first element is the XmlHttpRequest object, which allows the browser to communicate to a back-end server without performing a full postback. The next piece is JavaScript, which provides the capabilities to communicate with the back-end server. It's role is vital because at moment it is the only client script language that is supported across all browsers. Document Object Model (DOM) is the support to dynamically update form elements. This allows developers to write a single piece of code that targets multiple browsers. Finally, data transport with XML and JSON (JavaScript Object Notation) allows for the ability to communicate with the web server in a standardized way [8].

3 Atlas

Atlas builds on the .NET Framework 2.0 and adds support for better utilizing the capabilities of client-side JavaScript and the XMLHttpRequest object. It includes server-based features that make it easy to enrich existing ASP.NET applications, as well as a client script library that is used by the Atlas controls and services. The Atlas architecture extends across client and server and should be viewed as a broad set of development technologies for creating richer, more responsive cross-browser Web applications [4].

3.1 Atlas Architecture

The first major achievement of the Atlas architecture is that it spans both the client and the server. ASP.NET 2.0 added some additional client features, but not to the extent that Atlas does. Atlas server features are built on top of ASP.NET 2.0 and extend its capabilities. Atlas includes a new set of server controls as well as new features aimed at accessing server-based data and services from the browser [6].

The Atlas Client Script Framework is an extensible, object-oriented completely JavaScript client framework that allows you to easily build AJAX-style browser applications with rich UI and connectivity to web services. With Atlas, you can write web applications that use a lot of DHTML, JavaScript, and XMLHTTP, without having to be an expert in any of these technologies. It is the foundation on the client that is used heavily by the new Atlas features to enable richer application development with improved client-server interactions [7].

Client-server interaction with Atlas moves away from complete page refreshes. Instead, the initial HTML is retrieved and subsequent calls to the server get updated data in XML, JavaScript Object Notation (JSON), or snippets of HTML to update the page incrementally. The background asynchronous calls can invoke Web services or retrieve page changes without leaving users feeling that they must pause. These asynchronous calls manage updated view state information for subsequent server postbacks so that when a complete page refresh is necessary, the accurate state of the page is carried to the server [4].

3.2 The Client Script Core Library

At the heart of Atlas is the Atlas Client Script Framework, an extensible, object-oriented client framework that is itself based completely on JavaScript. This Framework takes most of the detail work out of writing web applications that are loaded with DHTML and JavaScript. It works on all modern browsers and is server-agnostic. No client installation is required [5].

The Atlas client script library is delivered to the browser as several distinct pieces. The script core comprises the bottom layers on which the rest of the library is built. There is also a browser compatibility layer. A key feature of Atlas is that it will run on the modern browsers that support the key elements of AJAX. The type of browser making a request automatically determines what browser-specific parts of the compatibility layer are used. The higher-level code has been written to the abstraction layer, so you do not need to code around variations in browser implementations [5].

On top of the compatibility layer is the core type system. The type system allows for an object-oriented approach to JavaScript development. It gives the developer working in JavaScript the ability to create namespaces and add classes to them. It also simulates object inheritance. There is support for interfaces, delegates, and enumerations, which makes it easier to switch back and forth between developing code on the server in an object-oriented programming language like C# and writing JavaScript code on the client [5].

The base class library layer built on top of the type system completes the core of the client script library. There is an Event object that supports multicasting of events in a natural way in JavaScript. There is a StringBuilder object, too. There is also support for object serialization, including support for JSON and XML data. The base class library also includes WebRequest and WebResponse classes that provide an abstraction over the browser's XMLHttpRequest object similar to those found in the System.Net namespace in the .NET Framework [5].

Atlas also includes integrated support for web services and debugging. By default, JavaScript running within a web browser does not know how to communicate with a web service. The Atlas Client Script Framework permits easy access to any ASP.Net-hosted web service, with proxy generation and object serialization all handled automatically. Support is also included for the specific building-block services that are part of ASP.Net 2.0. These services provide often-needed functionality including user profile storage, authentication, and user interface personalization [5]. Another nice aspect of the current Atlas installation is that debug versions of the client script library are included to make debugging and troubleshooting easier. Debugging JavaScript has always been cumbersome, so this may ease the pain [4].

4 User Controls

With the Atlas framework, there are user controls that are included that allow the developer to integrate the Atlas functionality into the page. These controls work like other ASP.NET server controls. They provide the HTML necessary to communicate back to the server from the client.

4.1 Script Manager

The ScriptManager control is central to Microsoft ASP.NET 2.0 AJAX Extensions, and must be included on all pages that will use Atlas features. It manages all ASP.NET AJAX resources on a page including providing core Microsoft AJAX Library resources to the client and coordinating partial-page updates using UpdatePanel controls [1]. Server controls can provide JavaScript for the client and utilize the ScriptManager control for handling it. The ScriptManager control uses the new IScriptComponent interface that the control implements. There must be only one ScriptManager on a Web page and at least one UpdatePanel for Partial-Page rendering to function. There can be multiple UpdatePanels on a page and they can also be nested within each other. The ScriptManager then controls the JavaScript generated from the controls within the UpdatePanel that is sent back to the server. This way only the UpdatePanel JavaScript is sent back to the server instead of the entire page.

4.2 Update Panel

UpdatePanel controls are a central part of Microsoft ASP.NET AJAX. They are used with the ScriptManager control to enable partial-page rendering. Partial-page rendering reduces the need for synchronous postbacks and complete page refreshes when only part of the page needs to be updated. Partial-page rendering improves the end user experience because it reduces the screen flicker that occurs during a full page postback and improves Web page interactivity. If the UpdateMode property of the UpdatePanel control is set to Always, the UpdatePanel control's contents are updated on every postback that originates from the page. This includes asynchronous postbacks from controls that are inside other UpdatePanel controls and postbacks from controls that are not inside UpdatePanel controls [2].

If the UpdateMode property is set to Conditional, the UpdatePanel control's contents are updated when one of the following occurs [2]:

- You call the Update() method of the UpdatePanel control explicitly.

- The UpdatePanel control is nested inside of another UpdatePanel control and the parent panel is updated.

- A postback is caused by a control that is defined as a trigger using the Triggers property of the UpdatePanel control. In this scenario, the control explicitly triggers a refresh of the panel content. The control can be either inside or outside the UpdatePanel control that the trigger is associated with.

- The ChildrenAsTriggers property is set to true and a child control of the UpdatePanel control causes a postback. Child controls of nested UpdatePanel controls do not cause an update to the outer UpdatePanel control unless they are explicitly defined as a trigger.

You can use multiple UpdatePanel controls to specify different page regions to be updated independently. When the page that contains one or more UpdatePanel controls is first rendered, all the contents of each UpdatePanel control are rendered and sent to the client. On subsequent asynchronous postbacks, each UpdatePanel control's contents might or might not be updated depending on the panel settings and on client or server logic specific to the panel [2].

4.3 UpdateProgress

The UpdateProgress control enables you to provide feedback on the progress of partial-page rendering. For postbacks or initial page rendering, UpdateProgress control content is not displayed. There can be many UpdateProgress controls on a page, each associated with a different UpdatePanel control. Alternatively, you can use one UpdateProgress control and associate it with all UpdatePanel controls on the page. The UpdateProgress control renders a <div> element that is displayed or hidden depending on where a postback originates and on whether the AssociatedUpdatePanelID property of the UpdateProgress is set. You associate UpdateProgress controls with an UpdatePanel control by setting the AssociatedUpdatePanelID property of the UpdateProgress control. When a postback event originates from inside an UpdatePanel control any associated UpdateProgress controls are displayed. If you do not set the AssociatedUpdatePanelID property, the UpdateProgress control will display progress for any asynchronous postback that originates from inside any UpdatePanel or for controls that are triggers for panels [3].

5 Converting the Dashboard into AJAX

Converting the current dashboard in to AJAX is a very simple process. The first thing that we have to do is add a tag mapping element to the web.config file. We are using the Atlas WebPartManager instead of using the previously used ASP.Net 2.0 WebPartManager. We also do this for the WebPartZone. We do this to enable the Atlas capabilities of WebParts.

```
<tagMapping>
   <add tagType="System.Web.UI.WebControls.WebParts.WebPartManager"
mappedTagType="Microsoft.Web.UI.Controls.WebParts.WebPartManager"/>
   <add tagType="System.Web.UI.WebControls.WebParts.WebPartZone"
mappedTagType="Microsoft.Web.UI.Controls.WebParts.WebPartZone"/>
</tagMapping>
```

The next step is to add the ScriptManager to the web page. Then we will enable partial rendering. Finally you need to add the Update Panel. Within the UpdatePanel we placed the ContentPlaceHolder. Now we have just implemented our first iteration of Ajax. There are many other things that we are going to pursue, but this is just the beginning.

6 Challenges

The process of converting to AJAX the **Healthcare Provider's Dashboard** application did pose many challenges. This new technology has not been standardized, but instead there are many people who are working just to figure out the best way to implement AJAX. We found that the best way to work with AJAX was to just by trial and error. Our whole project was actually seen as one big trial and error, posing many challenges and also satisfaction gained from figuring out the solutions to each challenge.

6.1 Challenge 1

The main challenge we ran into was that the AJAX ("Atlas") UpdatePanel control does not support Web Parts like it should. Downloading version 1.0 from http://ajax.asp.net/ installs AJAX onto your computer and allows you to create AJAX enabled Web sites. However if one wants to create a Web site using Web Parts they will not work correctly. By putting the Web Part Zones inside an UpdatePanel the Web Parts should be editable and drag-and-droppable with partial rendering. Thus, the page will not flicker for either action. Although, any action or click that is done with a Web Part the WebPartManger DisplayMode is always reset to BrowseMode (the default mode), which does not allow drag-and-drop or editing. Therefore, only one click works like it should before the DisplayMode is reset. This was a big issue in this project because the Provider Dashboard is always set to be in EditMode, which supports editing and drag-and-dropping at all times.

6.2 Solution to Challenge 1

The bright side to challenge 1 was that we were able to solve this problem. The solution is actually quite simple. We discovered that the Microsoft.Web.Atlas.DLL included its very own ScriptManager, UpdatePanel, WebPartManger, and WebPartZone controls. This DLL comes with the AJAX download but is not installed in Visual Studio by default. In order to use these controls, simply add them to the toolbox. This can be done by right clicking on the toolbox ➔ Choose Items and then browse for Microsoft.Web.Atlas.DLL. By using these controls and putting the WebPartZones inside this new UpdatePanel the problem was solved. Now we were able to edit, drag-and-drop, and do any action within a WebPart without a full postback.

6.3 Other Challenges

Consequently, the solution to challenge 1 opened up another "can of worms". By using the ScriptManager from the Atlas.DLL we were unable to use the controls that come in the AJAX Toolkit because the controls look for the original AJAX DLL; the one that is installed with the version 1.0 download. So we had to decide which was more important; partial rendering with WebPart functionality or full postbacks when editing or drag-and-dropping WebParts, but having the capabilities of AJAX controls from the AJAX Toolkit.

On the Provider Dashboard we have decided to use the Atlas.DLL to allow partial rendering when working with the Web Parts since the majority of the clicks on the page will be within the Web Part zones. Consequently many aesthetic features that we would have liked to accomplish with AJAX had to be done outside of the Atlas framework. We will continue to pursue these many challenges one step at a time in order to accomplish a crisp and clean project.

7 Conclusions

There were many problems with our first implementation of the Provider Dashboard. We saw that the page seemed to have lag time, and did not provide an optimal experience for the user. So instead of starting from scratch we optimized our current dashboard by implementing Ajax. The idea that we are trying to pursue with Ajax is a dashboard that will have no lag time.

We achieved what we wanted with Atlas. This platform allowed us to optimize the current application, and create a much faster, cleaner dashboard. The technology used, though brand new and not nearly flawless, definitely was exactly what the application needed.

References

[1] “Script Manager”, *Ajax. 2006. Microsoft Corporation.*, <http://ajax.asp.net/docs/mref/T_System_Web_UI_ScriptManager.aspx>

[2] “Update Panel”, *Ajax. 2006. Microsoft Corporation*., <http://ajax.asp.net/docs/mref/T_System_Web_UI_UpdatePanel.aspx>

[3] “Update Progress”, *Ajax. 2006. Microsoft Corporation.*, <http://ajax.asp.net/docs/mref/T_System_Web_UI_UpdateProgress.aspx>

[4] Gibbs, Matt “Atlas At Last ”, *MSDN. 2007. Microsoft Corporation*, July 2006, <http://msdn.microsoft.com/msdnmag/issues/06/07/AtlasAtLast /default.aspx#contents>

[5] Aitken, Peter “Ajax, Atlas, and all that Stuff”, *DevSource. 2007*, Ziff Davis Media Inc., February 26, 2006, <http://www.devsource.com/print_article2 /0,1217,a=172234,00.asp>

[6] Guthrie, Scott “Atlas Project”, *ScottGu's Blog*, June 28, 2005, <http://weblogs.asp.net/scottgu/archive/2005/06/28/416185.aspx>

[7] “Ajax” March 5, 2007, <http://en.wikipedia.org/wiki/AJAX>

[8] McClure, B. Wallace “Beginning Ajax with ASP.NET”, *Wiley Publishing*. Inc., 2007.