Syed.Asadullah Hussaini, S.Nasira Tabassum, M.Khader Baig / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 3, Issue 2, March -April 2013, pp.111-117

# Ajax Architecture Implementation Techniques

**Syed.Asadullah Hussaini, S.Nasira Tabassum, M.Khader Baig**
*Master of Technology, Shadan College, Affiliated to JNTU Hyderabad, AP .India
**Master of Technology, Nizam College, Affiliated to JNTU Hyderabad,A.P.India
***Master of Technology, Nizam College, Affiliated to JNTU Hyderabad,A.P.India

## ABSTRACT
**Today's rich Web applications use a mix of Java Script and asynchronous communication with the application server. This mechanism is also known as Ajax: Asynchronous JavaScript and XML. The intent of Ajax is to exchange small pieces of data between the browser and the application server, and in doing so, use partial page refresh instead of reloading the entire Web page. AJAX (Asynchronous JavaScript and XML) is a powerful Web development model for browser-based Web applications. Technologies that form the AJAX model, such as XML, JavaScript, HTTP, and XHTML, are individually widely used and well known. However, AJAX combines these technologies to let Web pages retrieve small amounts of data from the server without having to reload the entire page. This capability makes Web pages more interactive and lets them behave like local applications. Web 2.0 enabled by the Ajax architecture has given rise to a new level of user interactivity through web browsers. Many new and extremely popular Web applications have been introduced such as Google Maps, Google Docs, Flickr, and so on. Ajax Toolkits such as Dojo allow web developers to build Web 2.0 applications quickly and with little effort.**

*Keywords* - **Web applications, Java Script, Web application 2.0, Ajax architecture technology**

## INTRODUCTION

Ajax, which consists of HTML, JavaScript™ technology, DHTML, and DOM, is an outstanding approach that helps you transform clunky Web interfaces into interactive Ajax applications. Ajax is shorthand for Asynchronous JavaScript and XML (and DHTML, and so on). The phrase was coined by Jesse James Garrett of Adaptive Path and is, according to Jesse, not meant to be an acronym.

AJAX is a technique for creating fast and dynamic web pages. AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page. AJAX applications are browser and platform independent!

AJAX is based on internet standards, and uses a combination of:

- XMLHttpRequest object (to exchange data asynchronously with a server)
- JavaScript/DOM (to display/interact with the information)
- CSS (to style the data)
- XML (often used as the format for transferring data)

## Why AJAX?

AJAX allows feature-rich, dynamic web applications which use server-side processing without requiring the traditional "submit data — retrieve web page" methodology.

Using XML Http Request, data is transmitted behind the scenes of your web application and JavaScript is used to manipulate the application interface and display dynamic information.

This allows more streamlined applications that require less processing and data transmission because entire web pages do not need to be generated for each change that occurs. Instead, one web application reflects all of the changes that occur. JavaScript can also be used to allow higher levels of interactivity than allowed through HTML itself (e.g., keyboard shortcuts, click and drag, etc.

## Why Not AJAX?

AJAX will not work in all web browsers. As its name suggests, AJAX requires JavaScript. This alone means that AJAX applications will not work in web browsers and devices that do not support JavaScript. For this reason it is not accessible to many typical Web users.

The Web Content Accessibility Guidelines external link also require that web applications function when JavaScript is disabled or not

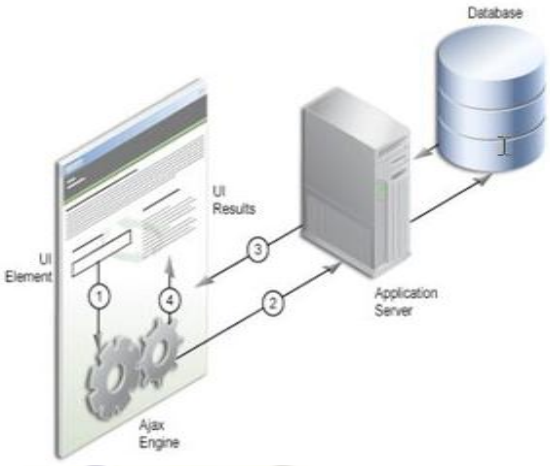
supported. AJAX also requires that XML Http Request be supported, which many browsers do not. The current solution to these problems is to either provide a non-AJAX alternative to your application or to allow your AJAX application to continue to function if JavaScript and XML Http Request are not supported. Such a requirement may be very difficult to achieve.

While developers may choose to require the users to use a browser that supports AJAX, they must understand that such requirements may not be possible for all users - especially those using portable devices or older web browsers. By its nature, AJAX tends to update and manipulate interface elements 'on the fly'.

AJAX also can submit information to the server without user interaction or may do so in methods that are not obvious to the user. For example, most users expect forms to be submitted, validated, and processed when a submit button is selected, but with AJAX this submission and processing can occur at any time (e.g., every 5 seconds, when a form element loses focus, etc.). It may not be apparent to users that information is being processed and saved - and this confusion can be intensified by the fact that AJAX can perform these operations very quickly. Most users expect there to be some delay before feedback or additional information is presented and typically expect the entire page to refresh indicating a new display - with AJAX, none of these visual cues may be apparent.

Another issue with AJAX is how the application interface is updated. When updates to the interface occur, it may not be visually apparent that a change has occurred. The problem is even more troublesome for screen reader users. Screen readers typically read in a linear fashion. When changes happen in the interface, the screen reader user may not be aware of the change and the new content will likely not be read.

In short, to allow dynamic interface changes to be accessible, the application must alert the user that a change has occurred, allow direct access to the new content, and then allow continued functionality of the web application. This process, while difficult to achieve, especially for screen reader users, is possible to achieve in many AJAX applications.




## Web Page as Application

Ajax blurs the boundary between web pages and applications. In classic web applications,a web page is an HTML document that can be rendered by a browser for information display purpose. It has limited or often zero intelligence on its own.

In an Ajax application, the HTML page the server sends to the browser includes code that allows the page to be a lot "smarter". This code runs in the background acting as the "brain" while the HTML document is rendered in the browser window. The code can detect events such as key strokes or mouse clicks and perform actions responding to these events, without making a round trip to the server.

Through Ajax, a web page feels like a desktop application. It responds fast, almost immediately to user actions, without full page refresh. It can further continuously update the page by asynchronously fetching data from the server in the background, achieving desktop application experience.

## AJaX Application Architecture

Given the challenges associated with Ajax, it is particularly important to architect an Ajax application properly. Otherwise the result can be either lackluster performance or code maintenance nightmare, or even both. There are two items impact Ajax application architecture significantly: the choice of an Ajax engine and client-side application logic implementation.

**AJAX ENGINE:** From the point of view of software architecture, the significant difference between an Ajax application and a classic HTML web application is the introduction of a client-side

engine. This engine, which runs inside the Web browser, acts as an intermediary between the applications's UI and the server. User activity leads to calls to the client-side engine instead of a page request to the server. Likewise, data transfer takes place between the server to the client-side engine, rather than directly to the Web browser.

Ajax engine is the key to the AJAX application model. Without it, every event generated by user activity must go back to the server for processing. There are many different ways to implement the client side Ajax engine. One approach is to write it from scratch based on the application need. Another approach is to use an Ajax toolkit that is available in the market today. There are many Ajax toolkits today, a lot of which are open source. Some toolkits are communication libraries, some of them are rich user interface components and some of them provide both. Choosing the right toolkit would significantly lower application development and maintenance challenge.

**WEB APPLICATION:** The core of a Web application is its server-side logic. The Web application layer itself can be comprised of many distinct layers. The typical example is a three-layered architecture comprised of presentation, business, and data layers. Figure illustrates a common Web application architecture with common components grouped by different areas of concern.

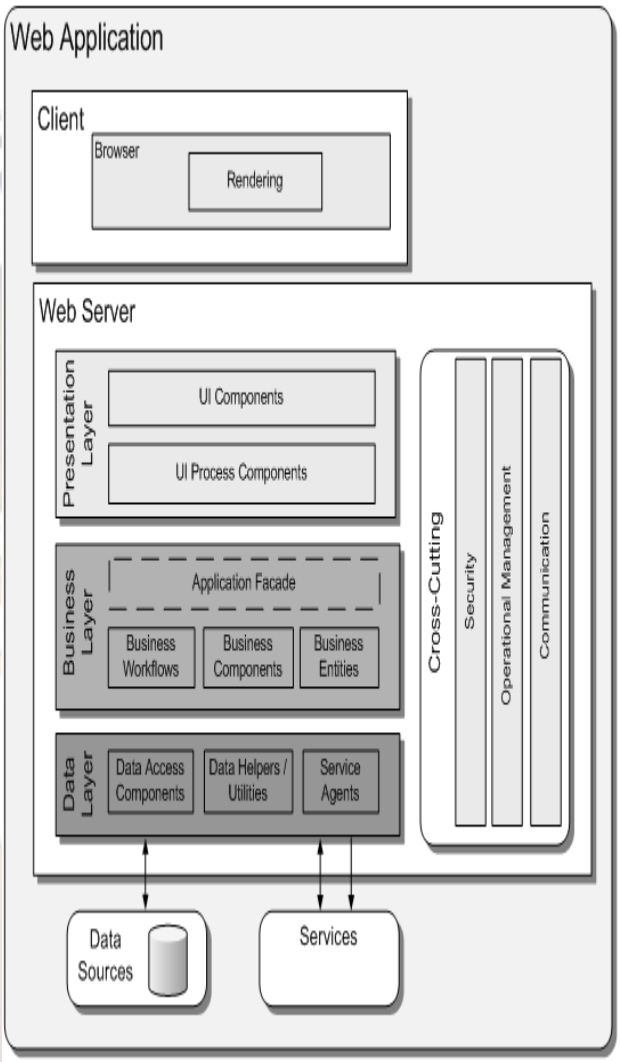
## Design Considerations

When designing a Web application, the goals of a software architect are to minimize the complexity by separating tasks into different areas of concern while designing a secure, high-performance application.

When designing your Web application, consider the following guidelines:

- **Partition your application logically.** Use layering to partition your application logically into presentation, business, and data access layers. This helps you to create maintainable code and allows you to monitor and optimize the performance of each layer separately. A clear logical separation also offers more choices for scaling your application.

- **Use abstraction to implement loose coupling between layers.** This can be accomplished by defining interface components, such as a façade with well-known inputs and outputs that translates requests into a format understood by components within the layer. In addition, you can also use Interface types or abstract base classes to define a shared abstraction that must be implemented by interface components.




- **Understand how components will communicate with each other.** This requires an understanding of the deployment scenarios your application must support. You must determine if communication across physical boundaries or process boundaries should be supported, or if all components will run within the same process.

- **Reduce round trips.** When designing a Web application, consider using techniques such as caching and output buffering to reduce round trips between the browser and the Web server, and between the Web server and downstream servers.

- **Consider using caching.** A well-designed caching strategy is probably the single most important performance-related design consideration. ASP.NET caching features include output caching, partial page caching, and the cache API. Design your application to take advantage of these features.

- **Consider using logging and instrumentation.** You should audit and log activities across the layers and tiers of your application. These logs can be used to detect suspicious activity, which frequently provides early indications of an attack on the system.

- **Avoid blocking during long-running tasks.** If you have long-running or blocking operations, consider using an asynchronous approach to allow the Web server to process other incoming requests.

- **Consider authenticating users across trust boundaries.** You should design your application to authenticate users whenever they cross a trust boundary; for example, when accessing a remote business layer from your presentation layer.

- **Do not pass sensitive data in plain text across the network.** Whenever you need to pass sensitive data such as a password or authentication cookie across the network, consider encrypting and signing the data or using Secure Sockets Layer (SSL) encryption.

- **Design your Web application to run using a least-privileged account.** If an attacker manages to take control of a process, the process identity should have restricted access to the file system and other system resources in order to limit the possible damage.

## Web Application Frame

There are several common issues that you must consider as you develop your design. These issues can be categorized into specific areas of the design. The following table lists the common issues for each category where mistakes are most often made.

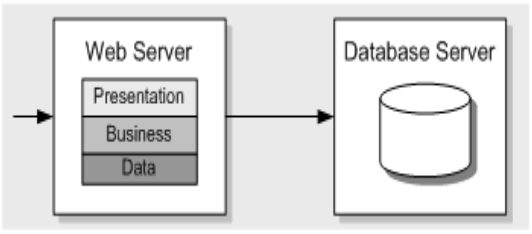
**APPLICATION LOGIC PARTITION:** Regardless of the client-side Ajax engine is implemented, how to partition application logic directly impacts application performance and maintainability. "Application logic partition" refers to the amount of application logic that runs on the client side versus the amount of logic that runs on the server side.

On the one side, putting more logic on the client side would deliver better application performance. However, client-side logic can easily result in a lot of hard to maintain JavaScript code. For example, Google Map is a relatively simple application and has limited functionality, but it still has more than a hundred Kilobytes of JavaScript logic on the client side (after obfuscation and compression). On the other side, putting more logic on the client side can potentially create application maintenance problem that is expensive and hard to scale to large development teams. What kind of logic should be put on the client side, how much logic and how the logic should be implemented? These are key questions developers must evaluate carefully in order to build manageable and maintainable applications. Ajax development model offers a lot of flexibility in application logic partition. This is a client-centric model that resembles closely to the typical desktop application model. This is a server-centric model that is very similar to the classic HTML web application model except for the "RIA" Ajax engine on the client side.

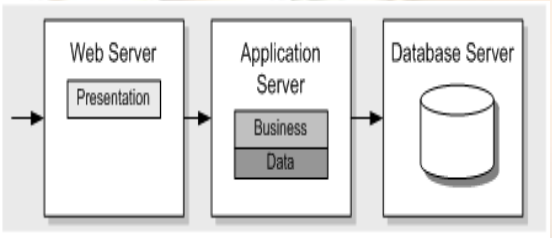
Obviously, developers can decide to partition their application anywhere between these two extreme cases. What is worthy of pointing out is that the server-centric model is fully capable of delivering a rich user experience such as rich UI and asynchronous partial updates. The reason is the introduction of the RIA Ajax engine.

In this model, the number of round trips is not necessarily reduced comparing with the classic HTML application model, but the amount of data to be transferred is much smaller. The asynchronous nature of the Ajax engine still enables the "continuous" user experience. The popular JavaServer Faces (JSF) model is such a server-centric model that encourages all processing happening on the server side. The benefits of this model include not only much enhanced user experience than a classic HTML application by the introduction of a client-side Ajax engine, but also good application maintainability. Because all logic stays on the server side, it is much easier develop and maintain application code on the server side than dealing with JavaScript code on the client side.

**NON-DISTRIBUTED DEPLOYMENT:** In a non-distributed deployment scenario, all the logically separate layers of the Web application are physically located on the same Web server, except for the database. You must consider how the application will handle multiple concurrent users, and how to secure the layers that reside on the same server. Figure below shows this scenario.

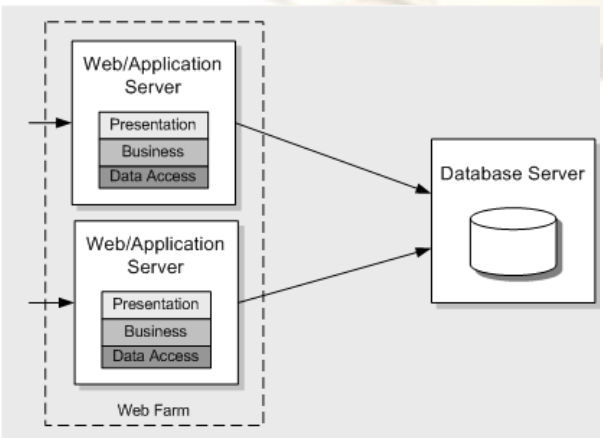



**DISTRIBUTED DEPLOYMENT:** In a distributed deployment scenario, the presentation and business layers of the Web application reside on separate physical tiers, and communicate remotely. You will typically locate your business and data access layers on the same sever. Figure below shows this scenario.




## Load Balancing

When you deploy your Web application on multiple servers, you can use load balancing to distribute requests so that they are handled by different Web servers. This helps to maximize response times, resource utilization, and throughput. Figure below shows this scenario.




**LOAD BALANCING A WEB APPLICATION:**

Consider the following guidelines when designing your Web application to use load balancing:

- Avoid server affinity when designing scalable Web applications. Server affinity occurs when all requests from a particular client must be handled by the same server. It usually occurs when you use locally updatable caches, or in-process or local session state stores.
- Consider designing stateless components for your Web application; for example, a Web front end that has no in-process state and no stateful business components.
- Consider using Windows Network Load Balancing (NLB) as a software solution to implement redirection of requests to the servers in an application farm.

## WEB FARM CONSIDERATION

A Web farm allows you to scale out your application, which can also minimize the impact of hardware failures. When you add more servers, you can use either a load-balancing or clustering approach.

Consider the following guidelines when designing your Web application to use a Web farm:

- Consider using clustering to minimize the impact of hardware failures.
- Consider partitioning your database across multiple database servers if your application has high input/output requirements.
- Consider configuring the Web farm to route all requests from the same user to the same server in order to provide affinity where this is required.
- Do not use in-process session management in a Web farm when requests from the same user cannot be guaranteed to be routed to the same server. Use an out-of-process state server service or a database server for this scenario.

## Technology Considerations

On the Microsoft platform, from an ASP.NET standpoint, you can combine the ASP.NET Web Forms model with a range of other technologies,

including ASP.NET AJAX, ASP.NET MVC, Microsoft Silverlight™, and ASP.NET Dynamic Data.

Consider the following guidelines:

- If you want to build applications that are accessed through a Web browser or specialized user agent, consider using ASP.NET.
- If you want to build applications that provide increased interactivity and background processing, with fewer page reloads, consider using ASP.NET with AJAX.
- If you want to build applications that include rich media content and interactivity, consider using ASP.NET with Silverlight controls.
- If you are using ASP.NET, consider using master pages to implement a consistent UI across all pages.
- If you are building a data-driven Web application with pages based on the data model of the underlying database, consider using ASP.NET Dynamic Data.

## TECHNOLOGIES

The term Ajax has come to represent a broad group of web technologies that can be used to implement a web application that communicates with a server in the background, without interfering with the current state of the page. In the article that coined the term Ajax, Jesse James Garrett explained that the following technologies are incorporated:

- HTML (or XHTML) and CSS for presentation
- The Document Object Model (DOM) for dynamic display of and interaction with data
- XML for the interchange of data, and XSLT for its manipulation
- The XMLHttpRequest object for asynchronous communication
- JavaScript to bring these technologies together

Since then, however, there have been a number of developments in the technologies used in an Ajax application, and the definition of the term Ajax.

XML is not required for data interchange and therefore XSLT is not required for the manipulation of data. JavaScript Object Notation (JSON) is often used as an alternative format for data interchange, although other formats such as preformatted HTML or plain text can also be used.

## Conclusion

Ajax provides several architectural approaches, and each approach is supported by various commercial software products and/or open source projects. The architectural diversity provides IT managers and Web developers with the ability to choose the optimal architectural approach and best products in order to meet their particular requirements and fit in with their existing practices.

## References

1. Jesse James Garrett (18 February 2005). "Ajax: A New Approach to Web Applications". AdaptivePath.com. Retrieved 19 June 2008.

2. Ullman, Chris (March 2007). Beginning Ajax. wrox.ISBN 978-0-470-10675-4. Archived from the original on 5 July 2008. Retrieved 24 June 2008.

3. "Dynamic HTML and XML: The XMLHttpRequest Object". Apple Inc. Retrieved 25 June 2008.

4. Hopmann, Alex. "Story of XMLHTTP". Alex Hopmann's Blog. Retrieved 17 May 2010.

5. "A Brief History of Ajax". Aaron Swartz. 22 December 2005. Retrieved 4 August 2009.

6. "JavaScript Object Notation". Apache.org. Archived from the original on 16 June 2008. Retrieved 4 July 2008.

7. "Speed Up Your Ajax-based Apps with JSON". DevX.com.Archived from the original on 4 July 2008. Retrieved 4 July 2008.

8. "Why use Ajax?". InterAKT. 10 November 2005.Archived from the original on 29 May 2008. Retrieved 26 June 2008.

9. "Deep Linking for AJAX".

10. "HTML5 specification". Retrieved 21 October 2011.

11. Prokoph, Andreas (8 May 2007). "Help Web crawlers efficiently crawl your portal sites and Web sites". IBM. Retrieved 22 April 2009.

12. Quinsey, Peter. "User-proofing Ajax".

13. "WAI-ARIA Overview". http://www.w3.org/. Archivedfrom the original on 26 October 2010. Retrieved 21 October 2010.

14. Edwards, James (5 May 2006). "Ajax and Screenreaders: When Can it Work?". sitepoint.com. Retrieved 27 June 2008.

15. "Access Control for Cross-Site Requests". World Wide Web Consortium. Archived from the original on 14 May 2008. Retrieved 27 June 2008.

16. "Secure Cross-Domain Communication in the Browser". The Architecture Journal (MSDN). Archived from the original on 29 March 2010. Retrieved 27 April 2010.

17. Cuthbertson, Tim. "What is asynchronous programming, and why is it so damn awkward?". http://gfxmonk.net/. Retrieved 19 October 2010.

18. "Selenium documentation: Fetching a Page".http://seleniumhq.org/. Retrieved 6 October 2011.

## Profiles

Syed.Asadullah hussaini has received his Master of Technology in Computer Science Engineering from Shadan college of Engineering & Technology, affliated to JNTUH Hyderabad,A.P.India. (Email: syed.asadullah hussaini@ymail.com)

S.Nasira Tabassum has received her Master of Computer Application from Muffakham Jah College of Engineering and Technology, Affiliated to Osmania University, Hyderabad, AP India. She received her Master in Technology in Software Engineering from Nizam Institute of Engineering and Technology, Deshmukhi, Nalgonda Dist, Affiliated to JNTU, Hyderabad, AP India. (Email: nasira.tabassum@gmail.com).

M.KHADER BAIG has received his Master of Technology in Computer Science Engineering from Nizam Institute of Engineering and Technology, Deshmukhi, Nalgonda Dist, Affiliated to JNTU Hyderabad, AP India. (Email: mogalkhaderbaig@gmail.com