IMPLEMENTING SYSTEM ARCHITECTURE OF AJAX

[1]P.SUMANKUMAR, [2]N.V.NEERAJA, [3]C.REDDINEELIMA, [4]V.RAJYALAKSHMI

[1]Associate professor, [2,3,4]Assistant professor
Computer science and engineering,
Mother Theresa institute of engineering and technology, palamaner, India.

Abstract: **AJAX stands for Asynchronous JavaScript and XML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script. Ajax uses XHTML for content, CSS for presentation, along with Document Object Model and JavaScript for dynamic content display. Conventional web applications transmit information to and from the sever using synchronous requests. It means you fill out a form, hit submit, and get directed to a new page with new information from the server. With AJAX, when you hit submit, JavaScript will make a request to the server, interpret the results, and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server. XML is commonly used as the format for receiving server data, although any format, including plain text, can be used. AJAX is a web browser technology independent of web server software. A user can continue to use the application while the client program requests information from the server in the background. Intuitive and natural user interaction. Clicking is not required, mouse movement is a sufficient event trigger. Data-driven as opposed to page-driven**

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

I. 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?

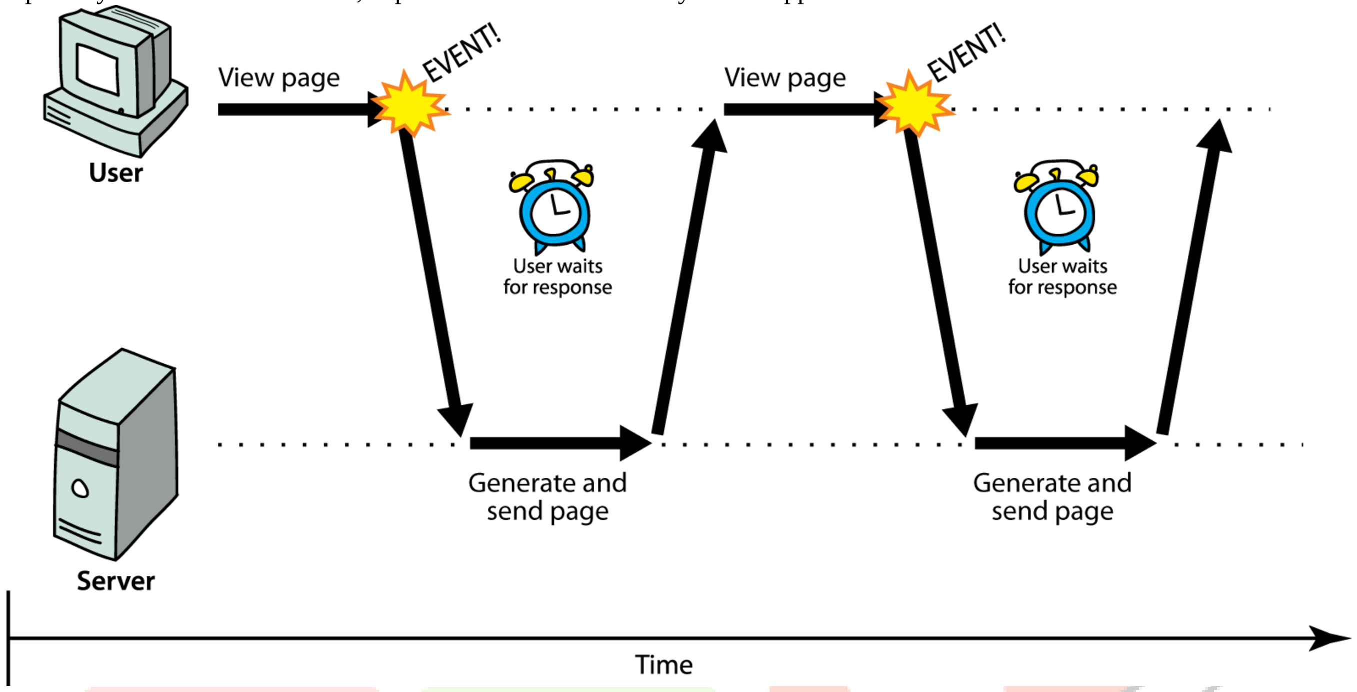
AJAX allows feature-rich, dynamic web applications which use server-side processing without requiring the traditional "submit data — retrieve web page" methodology. Using XML Http Request, data is transmitted behind the scenes of your web application and JavaScript is used to manipulate the application 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

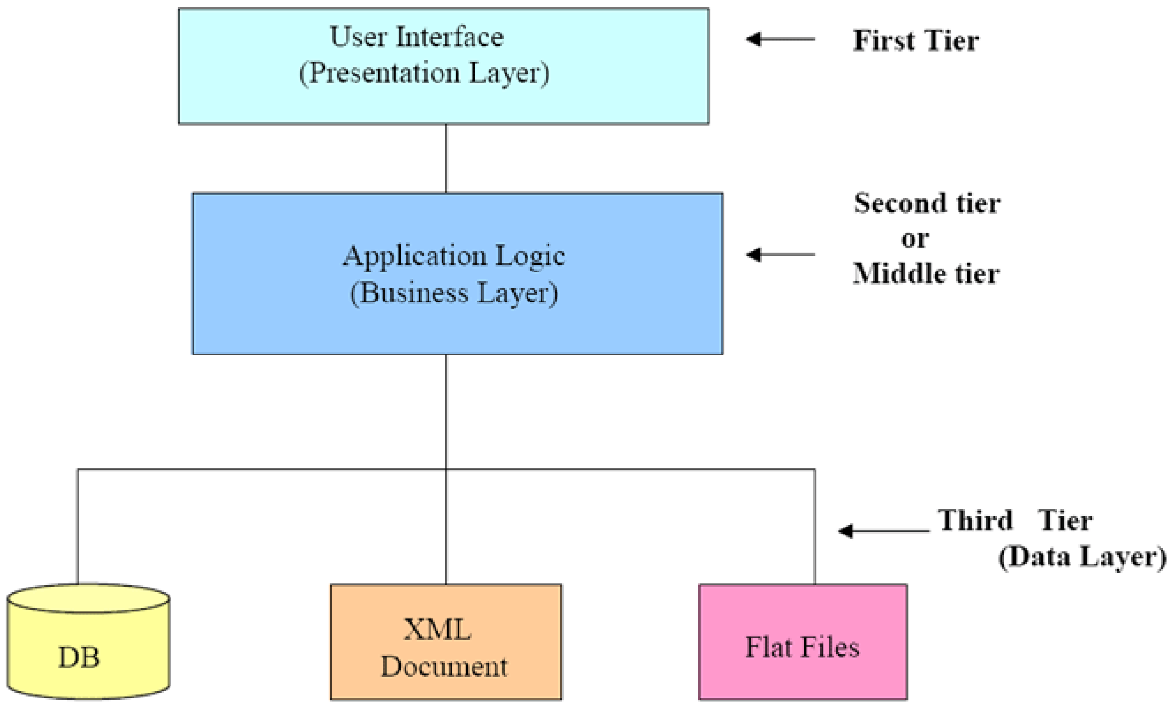
focus, etc.). It may not be apparent to users that information is being processed and saved - and this confusion can be intensified by the fact that AJAX can perform these operations very quickly.

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.



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.


User Interface (Presentation Layer)
First Tier
Application Logic (Business Layer)
Second tier or Middle tier
Third Tier (Data Layer)
DB
XML Document
Flat Files

Typical Three Tier Architecture

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.

Designing an AJAX Transaction

• Create an XMLHTTPRequest object
• Set up the request's onreadystatechange
function
• Open the request
• Send the request

Example Code (Client Side)

```
function sendRequest(textNode)
var xmlHttp = GetXmlHttpObject();
if (!xmlHttp) {
return false;
}
xmlHttp.onreadystatechange = function() {
if (xmlHttp.readyState == 4) {
textNode.nodeValue =
xmlHttp.responseText;
}
}
var requestURI =
"http://greatbeyond.org/cgi-bin/request.cgi";
xmlHttp.open("GET", requestURI, true);
xmlHttp.send(null);
}
```

Example Code (Server Side)

And we might have the following request.cgi in the cgi-bin directory of greatbeyond.org

```
#!/usr/bin/perl
print("Content-type: text/plain\n\n");
print("57 channels and nuthin' on");
```

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.

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

CONCLUSION

AJAX is a web browser technology independent of web server software. A user can continue to use the application while the client program requests information from the server in the background. Intuitive and natural user interaction. Clicking is not required, mouse movement is a sufficient event trigger. Data-driven as opposed to page-driven

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. c. Hibbs. AJAX on Rails Presentation. US: Addison-Wesley, 2005, no. 7, pp: 81-130.

12. J. James, G. AJAX. "A New Approach to Web Applications", Adaptive Path, 2005

13. C. Gross. AJAX Pattern and Best Practice (Expert's Voice). US: Addison-Wesley, 2005

14. P. Ballard. Sams Teach AJAX in 21 Days. US: Addison-Wesley, 2006

15.K. Smith, Simplifyin. AJAX-Style Web Development.pdf. Microsoft, 2006

16. C. Gross. AJAX Patterns and Best Practices. 2007

IJCRT