News

HTML 5 Web Sockets vs. Comet and Ajax

Posted by **Dionysios G. Synodinos** on Dec 11, 2008 09:57 AM

Community **.NET**, **Ruby**, **Java** Topics **Rich Internet Apps** , **Web 2.0** Tags **AJAX**

During the mid-90's the World Wide Web was rapidly growing as the dominant way to distribute information. As browsers were becoming ubiquitous and users became accustomed to the whole experience, it was clear that the WWW could also serve an application platform that would potentially reach more users than any other platform in history. It was still early though since the accompanying standards (HTML, HTTP, etc.) were not originally designed for high interaction and rich user experience. Some of the original work on providing rich online applications, can be accredited to the Microsoft Exchange engineers team. Since '96 they had been using the IFrame element in order to provide an Outlook-like frond end to their mail server system. These early attempts were lagging in terms of responsiveness and over-all user experience, but they were a clear indication of the things that were to come. At 1998 when they started writing the web front-end for MS Exchange Server 2000, they developed XMLHTTP, a component that allowed for asynchronous communication with the server from a single web page. As the later sentence suggest, XMLHTTP had really no immediate ties to XML. It has been suggested by Alex Hopmann who was then a member of the team, that the only reason that they had to use the XML prefix is because IE 5 was getting ready for its 2nd beta and they had to ship it as part of the MSXML library.

This new technology was also implemented by the Mozilla Foundation as XMLHttpRequest (XHR) in 2002 for the first version of their browser, which later became Firefox. Although there were other vendors that tried to utilize the power of these new APIs, the remote scripting paradigm that they represented did not gain public awareness until Google stared deploying a series of next-generation services based on JavaScript and XHR. The first service was Google Maps that was first made known from the Google Blog on February 8 2005. After that and within just a few months XHR became one of the hottest issues in our industry. At that time nobody could really predict the revolution that XHR would be bring to the development of Web Applications, but it was eminent that it was going to change the way we thought about the WWW.

With the release of Kaazing Gateway, InfoQ discussed with Richard Smith, about the evolution of technologies like AJAX, Comet and the promising HTML 5 Web Sockets:

> Ajax provides a mild salve to the HTTP communication model by enabling clients to asynchronously poll for server-side events. By polling, server events can be queued and delivered to the browser on each poll interval, which emulated server initiated communications and provides real-time message delivery within the bounds of the poll interval. Thus, true real-time communications are simply not possible with Ajax as our only tool.
>
> Comet introduced an even greater departure from the HTTP communications model by enabling "push"-style of communications over HTTP. Comet defines several techniques that allow the server to send information to the browser without prompting from a client. With the help of an additional HTTP connection, Comet can even facilitate bi-directional communications over two HTTP connections. However, the trouble with Comet is its lack of standard implementation due the varying levels of support provided by browser vendors for XHR and iFrames—the two building blocks of Comet-style communications. In addition, there is a significant amount of overhead, both in terms of networking and development, to manage two connections for communications. These costs can introduce latency into Comet applications that limit the accuracy of the real-time communications they provide.
>
> HTML 5 WebSocket represents the next evolution of Comet and Ajax in an attempt to stand HTTP communications on its head. The HTML 5 WebSocket specification defines a single-socket full-duplex (or bi-directional) connection for pushing and pulling information between the browser and server. Thus, it avoids the connection and portability issues of Comet and provides a more efficient solution than Ajax polling. At present HTML 5 WebSockets is the predominant mechanism for facilitating full-duplex, real-time communications on the Web.

Richard feels that both AJAX and Comet approaches have several limitations:

> Attempting to simulate server initiated communications with Ajax requires polling schemes, which blindly check for updates irrespective of state changes in the application. The result is poor resource utilization on both the client and server, since CPU-cycles and memory are needlessly allocated to prematurely or belatedly detect updates on the server. Consequently, depending on the rate in which events are published on the server, traditional Ajax applications must constantly strike a balance between shorter and longer polling internals in an effort to improve the accuracy of individual requests. Furthermore, high polling frequencies result in increased network traffic and server demands, while low polling frequencies result in missed updates and the delivery of stale information. In either case, some added latency is incurred in message delivery. Moreover, short polling intervals are costly because to support such abbreviated server pings requires significant server resources to scale.
>
> Comet attempts to deliver “push” communications by maintaining a persistent connection or long-lived HTTP request between the server and the browser. This connection allows the server to send events, initiated by the client to the browser. Upstream requests can be issued by the browser to server are made over an additional HTTP connection. Thus, Comet can facilitate by directional communications over two HTTP connections. However, the maintenance of these two connections introduces significant overhead in terms of resource consumption on the server because it takes double the resources to serve a single client. In addition, browsers are often configured to limit HTTP connections by domain. This further complicates the use of Comet and often requires complex techniques such as multiplexing or managing multiple domains.
>
> Some Comet solutions attempt to mitigate resource consumption by employing a technique known as long-polling. However, this technique sends an undue amount of HTTP request/response headers. For example, each time an event is sent by the server, the server severs its connection with the client browser, forcing the browser to reestablish its connection with the server. This action causes another client request and server response to be sent across the wire for each interval of the long-poll. More often than not the HTTP Headers in the response outweigh the message being delivered:

```
From client (browser) to server:
GET /long-polling HTTP/1.1\r\n
Host: www.kaazing.com\r\n
User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.9) Gecko/2008061017 Firefox/3.0\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
Keep-Alive: 300\r\n
Connection: keep-alive\r\n
Cache-Control: max-age=0\r\n
\r\n


From server to client (browser):
Date: Tue, 16 Aug 2008 00:00:00 GMT\r\n
Server: Apache/2.2.9 (Unix)\r\n
Content-Type: text/plain\r\n
Content-Length: 12\r\n
\r\n
Hello, world
```

> Development with Comet presents a number of challenges. It is possible to roll your own solution. This requires developing a JavaScript library that leverages a number of techniques such as forever frames and XHR Streaming to maintain a persistent connection. The trouble is the implementation of these techniques varies

from browser to browser, and to make matters worse they often depend on server-side code to forward snippets of JavaScript, which compounds the complexity of implementation as well as brings into question the subject of portability.

Fortunately, there are a few frameworks that simplify Comet development by providing abstractions for these transports. The most notable of which is the SitePen's Cometd, which is a reference implementation of the Bayuex specification—a spec that defines a publish-subscribe model for Comet. Recent versions of Jetty also include a Java-based server-side implementation of Bayeux.

Bayuex and Cometd definitely simplify Comet. However, there is a lot of controversy surrounding its APIs and wire protocol. Comet Daily provides an in depth look at the issues surrounding Bayuex in its "Coliding Comet: Battle of the Bayuex" series.

He suggests that the proposed HTML 5 WebSockets present many benefits over contemporary solutions:

Though Comet and Ajax can both deliver end-user experiences that provide desktop-like functionality and low user-perceived latency, only Web Sockets lives up to the promise of providing a native means to accurately and efficiently stream events to and from the browser with negligible latency. It is by far the most comprehensive solution for delivering real-time information over the Web. Not only does it provide full asynchronous duplex streaming communication with a single TCP/IP connection, but also benefits from few HTTP headers and more importantly allows the same message format to be used by both the browser and the origin service.

Most Comet implementations rely on the Bayeux protocol. The use of this protocol requires messages from the origin services to be transformed from the messages' initial format to conform to the Bayeux protocol. This transformation introduces unnecessary complexity in your system, requiring developers to manipulate one message format on the server (e.g. JMS, IMAP, XMPP, etc.) and a second message format (.e.g. Bayeux and JSON) on the client. Moreover, the transformation code used to bridge your origin protocol to Bayeux introduces an unnecessary performance overhead into your system by forcing a message to be interpreted and processed prior to being sent over the wire. With Web Sockets, the message sent by the server is the same message delivered to the browser, eliminating the complexity and performance concerns introduced by transformation code.

There is often the question of availability in regards to WebSockets. At the moment, browsers do not provide a native support. However, this is sure to change in the coming months as browsers such as WebKit, Firefox, and Opera have been historically quick to adopt other HTML 5 features such as canvas, postMessage, offline storage, and Server-sent events (SSE).

WebSockets also require some level of server-side support as there is an initial handshake over HTTP that is required in order to upgrade an existing HTTP connection to a raw TCP/IP connection. Kaazing Gateway, an open source project, represents the first server to provide this support as well as the scalability required to services tens of thousands of persistent connections. Kaazing, the vendor for Kaazing Gateway, also provides a JavaScript library that can enable any modern Web browser to take advantage of WebSockets. Therefore, WebSocket support is readily available today.

For working with HTML 5 WebSockets, Kaazing has released Kaazing Gateway 8.09_2 Atlantis which is an open source HTML 5 WebSocket Server that is available for use under the OSI approved Common Public Attribution License (CPAL) - a derivative of the Mozilla Public License:

Kaazing Gateway makes it possible for developers to take advantage of WebSockets today by providing a JavaScript library that emulates the HTML 5 WebSocket, making it possible to build applications that leverage the WebSocket interface and that can be deployed to both modern and future browsers.

The ultra high-performance server behind Kaazing Gateway can support tens of thousands of concurrent connections on a single node. Multiple instances can be clustered with traditional HTTP load-balancers or DNS round robin, making it possible to support any number of persistent client connections. In addition to large numbers of connections, Kaazing Gateway can also handle high data throughput thanks to its high-performance, staged event driven architecture (SEDA).

The Atlantis release of Kaazing Gateway also comes prepackaged with JavaScript clients for popular message services such as Apache ActiveMQ and RabbittMQ as well as clients for XMPP services such as OpenFire, Jabberd, and other popular chat servers. This makes it easy for you to quickly build web-based chat applications or messaging applications such as stock matrixes, online trading platforms, or online games.

The planned release the Kaazing Gateway 8.12 aims to deliver even more HTML 5 features such as Server-sent Events, more advanced security services, and extended support for XMPP (Jabber) and STOMP (e.g. ActiveMQ, RabbitMQ, or OpenMQ):

The libraries now enable any modern browser to support HTML 5 Server-sent Events, and introduce support for HTML 5 postMessage, which facilitates cross document messaging. Kaazing's HTML 5 libraries also include support for HTML 5 Offline Storage, which provides simple DOM-based storage solution. Kaazing Gateway and its client libraries also now provide support W3C Access Controls for Cross-Site Requests, which is a mechanism to enable client-side cross-site requests, better known as Cross-site XmlHttpRequests.

Beyond extended support for HTML 5, Kaazing Gateway 8.12 also provides more advanced XMPP features such as group chat. This release also introduces STOMP-JMS Adapter, which allows you to adapt any existing Java Message Service (JMS) services (e.g. JBoss Messaging, Tibco EMS, OpenMQ, SwiftMQ, WebSphere MQ, etc.) for use with Kaazing Gateway.

You can find more information on **AJAX**, **Comet** and **Rich Internet Applications**, right here on InfoQ.

- **This article is part of a featured topic series on** Rich Internet Apps
- See more Rich Internet Apps content at: http://www.infoq.com/ria
- Other recent content items in this topic
  - Java FX Technology Preview
  - Case study: Eclipse Rich Ajax Platform Use at CAS Software AG

4 comments **Reply**

Welcome HTML 5 by Fidel Chavarria Posted Dec 11, 2008 1:11 PM
WebSockets is a Comet technique by Dylan Schiemann Posted Dec 15, 2008 8:09 AM
Re: WebSockets is a Comet technique by Kacem Boufelliga Posted Jan 9, 2009 6:21 AM
One more note by Dylan Schiemann Posted Dec 15, 2008 8:43 AM

Sort by date descending

Welcome HTML 5

Dec 11, 2008 1:11 PM by **Fidel Chavarria**

This is great!
I´m glad to now that there are improvement ...

Reply

WebSockets is a Comet technique

Dec 15, 2008 8:09 AM by **Dylan Schiemann**

I think the article misses a key point, which is that WebSocket is simply a Comet technique, and that Comet is a term coined to encompass a variety of techniques and protocols to enable the real-time web.

Michael Carter and Orbited pioneered the WebSocket efforts and Kaazing followed their lead.

Users of Dojo can easily use either Cometd's Bayeux protocol that is widely adopted by companies such as IBM, Sun, and BEA, or they can use js.io with Dojo to get full WebSocket support: cometdaily.com/2008/10/29/dojo-and-websocket/

Others are choosing to mix and match. For example, Confetto uses both Bayeux and WebSocket by combining Orbited, Jetty, DWR, and Dojo.

Reply

One more note

Dec 15, 2008 8:43 AM by **Dylan Schiemann**

One more note... cometD is a Dojo Foundation project, not something solely created by SitePen.

Reply

Re: WebSockets is a Comet technique

Jan 9, 2009 6:21 AM by **Kacem Boufelliga**

I agree with Dylan.

Michael Carter of Orbited has specifically argued against the polling nature of Ajax and built his product around having a Web Socket. There is a great series of article/exchange about the Comet implementations, Bayeux in Dojo versus the Orbited way. It is worth reading all the parts
cometdaily.com/2008/02/07/colliding-comets-batt...

Reply