Next generation Push:

a viable technology for web-based FX market data delivery

Alessandro Alinone is CTO at Lightstreamer

Currently, one of the most utilized buzzwords regarding Internet technology is "Web 2.0", often associated with other new terms such as "Ajax" and "Comet". The goal of this article is to show how the maturity of these technologies finally permits the push of live FX market data via the Internet to any browser with low latency and high reliability.

The Web is changing under the pressure of its own success. In recent years more and more applications have been ported to a Web front-end, i.e. a "thin client" that runs entirely within a browser window without installing any external components. Until now the users have had to put up with poorer user interfaces, with respect to the ones offered by traditional desktop applications. The advent of Web 2.0, and Ajax in particular, is transforming the Web user's experience to make it very similar to a desktop application (i.e. interactive controls, fast response time, decoupling between user's actions and page loading, etc.). But this is still not enough for very sophisticated real-time applications, such as FX market data delivery.

To guarantee a very low latency between the generation of a price by the market and its presentation to the final user, a dedicated solution is necessary, namely "Push Technology". This term was coined in 1996 to refer to any technique addressed - to a greater or lesser degree and more or less effectively - to reverse the classic Web model.

The classic model (known as "pull") has the client (browser) solicit data from the server in a synchronous manner. This means that every time the client needs a data update, it has to ask the server expressly to find out if the data has changed and obtain the new value.

The push model, on the other hand, has the client receive updates in an asynchronous manner at the server's discretion, generally after expressing interest in a certain type of information (subscription phase). In other words, the client becomes a passive part of the system, receiving updated information as soon as it is available on the server, without having to ask for it every so often. In this sense, e-mail could be considered the oldest and most widespread form of push technology on the Internet.

Effective Push Technology

Early Push Technology did not prove to be the ideal solution for delivering live FX market data to a client through the Internet. The main three issues (that are now completely solved by the most mature and robust solutions) are:

- An inadequate control over used bandwidth and network congestion. Pushing real-time data can lead to unpredictable network traffic. This is because each user could subscribe to an arbitrary number of currency pairs and order monitors where the actual update frequency of each subscribed item is changeable. The solution to this is to use dynamic heuristic filtering algorithms to limit the bandwidth while keeping the overall data coherency. Furthermore some adaptive mechanisms can be employed to throttle the data flow based on the state of the network.

- Poor scalability on the server side. True push solutions need to maintain at least one open TCP socket for each connected client. Some Web/application servers have been extended to act as streaming engines, but their traditional architecture based on the "one-thread-per-connection" model makes scalability pretty impossible for tens of thousands of users. The new-generation streaming engines employ more suitable architectures. For example Lightstreamer, a comprehensive Web push solution, adopts a staged event-driven architecture instead of a thread-based one, making it possible to decouple the number of connections that the server can sustain from the number of threads that are employed.

- The necessity to download some external components onto the client PC (such as an application, an applet, a browser plug-in, etc.). This can now be substituted by a fully functional "zero-client install" solution, made feasible by combining several advanced Web techniques to create a paradigm that has existed for many years and has been recently named "Comet".

This last issue involves the perception of the final user more than the others, because a Web user is not accustomed to see live data on a common Web page displayed by a standard browser. To achieve such a goal, four paradigms were established.

Application Paradigm	Method of sending data with respect to the user's actions	Method of sending data with respect to the browser's actions
1. Traditional Web Application a.k.a. Page Refresh	Synchronous	Synchronous
2. Classic Ajax Application a.k.a. Periodic Polling	Asynchronous	Synchronous
3. Smart Ajax Application a.k.a. Smart Polling a.k.a. Asynchronous Polling a.k.a. Comet - Long Poll	Asynchronous	Partially Asynchronous
4. Streaming Ajax Application a.k.a. Streaming Ajax a.k.a. True Push/Streaming a.k.a. Comet - Forever Frame a.k.a. Reverse Ajax (this term is used for the Ajax polling paradigms too)	Asynchronous	Asynchronous

Figure 1. The different paradigms available to deliver real-time data through the Web

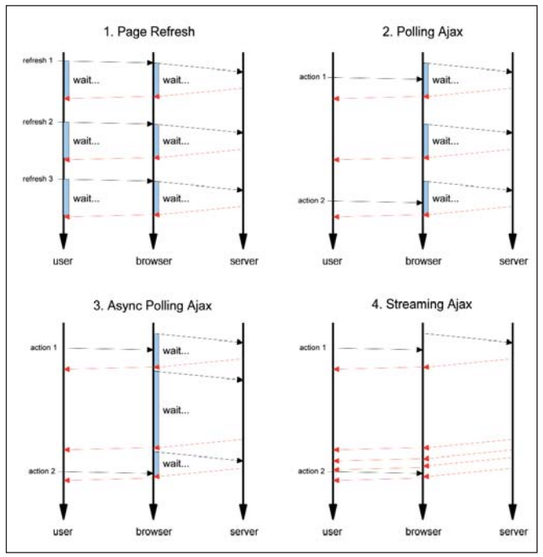

Figure 2. The four models used to implement a real-time Web

Figure 1 summarizes the alternative names by which the different paradigms are referred to. Figure 2 illustrates the interaction models related to the paradigms. In both the figures three tiers are used to explain the differences (user, browser, server), because the type of interaction that occurs between the server and the browser (i.e. the JavaScript engine) can be different from the interaction between the browser and its human user. In particular, a synchronous data delivery with respect to the user's actions means that to update the displayed data the user must take some action or that the user's actions are blocked during the update. A synchronous data delivery with respect to the browser's actions means that even if a user's action is not required, under the hood, the JavaScript code running inside the browser has to issue a request to the server and wait for a response for each update.

Traditional Web applications are based on page refreshes (automatic or user-driven reload action), in the course of which both the user and the browser are "blocked". In other words, the system is completely synchronous. Furthermore the update frequency that can be reached is very low.

Ajax applications have been introduced in order to avoid blocking the user's actions while retrieving data from the server. But in the classic Ajax model, based on periodic polling, the interaction between the browser and the server is still synchronous, resulting in a waste of bandwidth and in high latency for the data delivery.

To improve classic Ajax applications, the "asynchronous polling" technique has been introduced. In this case the polling period is not predefined, as in the two previous models, but is controlled by the server. If no fresh data is available, the server keeps the client request pending until new data is available. In this way a near real-time behavior is achieved, especially for low frequency events (higher frequency events still experiment a possible delay caused by a full request/response round trip that each update must undergo). Even when a true streaming model is implemented, the asynchronous polling technique should be kept as a backup for situations where some atypical proxy server blocks all the streaming traffic (but the switch between the two modes should be automatic, through some "stream-sense" feature).

Currently, the state-of-the-art paradigm is known as "streaming Ajax" or "Comet - forever frame", where true push/streaming is made possible on a very standard Web infrastructure. In this model the data delivery is fully asynchronous, both from the server to the browser and from the browser to the user's interface. This results in a very high update frequency, with low latency and low bandwidth, leading to an actual real-time system. This paradigm is implemented through a permanent connection from the browser to the server, on the top of which the server is able to deliver asynchronous messages adopting a publish/subscribe mechanism. When the browser receives an update through a JavaScript callback function, some code is executed to update the DOM (document object model) of the page in real time, in order to reflect the data change. Often some graphical effects are employed to catch the user's attention on the changing value. Some advanced frameworks make it even possible to plot a live streaming chart in the browser window by using only pure HTML and JavaScript.

Conclusion

There is no doubt that pushing live FX market data via the Internet can benefit all FX market participants (i.e. both sell-side and buy-side). Specific real-time dashboards for both the trading and sales roles can be developed, requiring on the client side nothing more than a very common Web browser and an Internet connection (not necessarily broadband). The result is an improvement in information pervasiveness and a simplification in the client-side system administration.