A Comparison of Delta-Communication Technologies and Techniques

NR Dissanayake[1#] and GKA Dias[1]

[1]*University of Colombo School of Computing, Colombo 7, Sri Lanka*

[#]nalakadmnr@gmail.com

Abstract— *Delta-Communication can be seen as the power of Rich Internet Applications, and there are different Techniques and Technologies available for the development of Delta-Communication, which should be selected carefully into the Rich Internet Application development. Enough discussions are not available, which compare and contrast these Delta-Communication development Techniques and Technologies towards supporting decision making of selecting them. This paper provides an overview of the contemporary Techniques and Technologies available for the Delta-Communication development, contextually compares them aligning to some selected criteria, and finally discusses some facts to be considered when selecting them for the Rich Internet Application development. A literature survey on the Delta-Communication development Technologies and Techniques was conducted, which was followed by a series of experiments towards getting the empirical evidence for the comparison. During the contextual comparison, the Simple Pull Delta-Communication was identified as the least complex technique and the Websocket was noted as the highest complex technology.*

Keywords— **Delta-Communication, Rich Internet Applications, Techniques and Technologies**

I. Introduction

Rich Internet Applications (RIAs) have become popular with their increased user experience delivered through rich GUIs and faster responses. The key concept behind the fast responses is the Delta-Communication (DC), which enables the communication of necessary data between the client and the server components, in either synchronous or asynchronous modes, supporting both data-pull and push modes (Dissanayake & Dias, 2017). There are various Technologies and Techniques (TTs) available for developing the DC in RIAs. These TTs have been already discussed in different forums, and also many tutorials are available to demonstrate the development of them. However, proper comparisons of these TTs are not available towards supporting decision making in selecting these TTs for the RIA development.

This paper gathers contemporarily used DC development TTs, provides an overview of their features, then compare and contrast them aligning to the facts: communication mode, complexity, ease of development, ease of maintenance and modifiability, and scalability. The paper also discusses some facts to be considered, when making decisions for selecting the DC TTs for the RIA development.

For this study, a literature survey was conducted focusing on identifying the available DC development TTs, their features, and development details. Parallel to the literature survey, a series of experiments was conducted towards getting empirical evidence on the identified DC development TTs. During the experiments, prototypes were developed using HTML, JavaScript, jQuery, PHP, and MySQL; and development process and the prototypes were examined focusing on the complexity, ease of development/maintenance/modifiability, and scalability factors.

Section II of this paper provides the background of the DC, stating some history and the definitions for the DC and related basics. Section III provides an overview of the contemporarily utilized DC TTs. Section IV delivers the contextualized comparison of the DC TTs discussing the results of the comparison. Based on the knowledge gained through the comparison, section V discusses some facts to be considered when selecting the DC TTs for the RIA development. Finally, section VI concludes the paper specifying the future applications of the knowledge delivered in this paper.

II. Background

This section provides some history of the DC concept, and also states definitions for the DC and related basics towards getting an adequate understanding of the concept of the DC before the comparison of the development TTs.

A. History of Delta-Communication

Microsoft was working on a technology named XMLHTTP in their Exchange 2000 project (Hopmann, n.d.), and it was first introduced to the world as an ActiveX control in Internet Explorer 5.0 in March 1999 (Dutta, 2006) (Smith, 2006); and later it was called the XMLHttpRequest (XHR) object, which has an Application Programme Interface (API) in JavaScript (JS).

In 2005, Jesse James Garrett from Adaptive Path coined the name AJAX, introducing the first JS based DC technique for the web applications, utilizing the XHR object (Garrett, 2005). This technique became popular and took the traditional web applications to a whole new era called Web2, which is the era of the RIAs. Later W3C acquired the control of the XHR object and released the first specification on 2006 (W3C, 2006). Since then the term AJAX has become another name for the RIAs, where even some developers refer the RIAs as AJAX applications.

AJAX can be seen as the beginning of the JS based RIA development approach, and it became a major breakthrough in the web development area (Salva & Laurencot, 2009). After its introduction, developers were learning how to use AJAX to create desktop-like GUIs in the web applications such as Google Maps; and later they subsequently used AJAX even to create entire enterprise RIAs (Lawton, 2008). Using the JS's ability to manipulate the Document Object Model (DOM) in HTML documents, AJAX achieves and enhances the interoperability capability of the web applications (Salva & Laurencot, 2009).

It should be noted that the AJAX itself is not a technology, it is a technique; and the technology behind AJAX is the XHR object with its JS API. The AJAX is a data-pull technique, employing the traditional request-response model. Combining HTML and CSS with JS, AJAX has become a powerful tool in RIA development, providing the fundamental implementation of the DC.

The main characteristics of AJAX and other DC TTs introduced after AJAX, are discussed in section III.

B. *Definitions for Delta-Communication and Related Basic Concepts*

Before discussing the DC TTs, it is important to understand what the DC is, its characteristics, and also related basic concepts.

The main features of the DC are the capability of processing in the background and then performing partial page rendering to display the results on the GUI (Dissanayake & Dias, 2017). The communication is done faster due to the smaller set of data communicated compared to the traditional web communications. Considering these facts, the DC is defined as: "*Delta-Communication is the rich communication model used by the rich features of the RIAs, for client-component(s) to communicate with the server-component(s), to exchange only the needful dataset – for a particular feature executed at the time – which is smaller, compared to the size of the request/response of traditional communication. Since the size of the dataset communicated is smaller, the communication completes faster, eliminating the work-wait pattern. The processing of the response is done by the client-components in the background, therefore the page refreshes are eliminated and replaced by partial page rendering to update the content of the GUI with the results of the response. The user experience can be determined by the implementation of the feature, in either blocking (synchronous) or non-blocking (asynchronous) modes"* (Dissanayake & Dias, 2017).

Simple Pull Delta-Communication (SPDC) is the simplest abstract implementation of the DC, and it is defined as: "*Simple Pull Delta-Communication is the basic abstract Delta-Communication technique, based on the data-pull mode. It describes the simplest form of data-pull Delta-Communication, based on the request-response model; and this technique is technology independent*" (Dissanayake & Dias, 2017).

Aligning to the definition of the SPDC, AJAX technique can be seen as JS implementation of the SPDC, which is limited to the browser based applications. The term AJAX expresses some flaws, and it is outdated with regards to the latest API version of the XHR object (Dissanayake & Dias, 2017). Due to the evolution of the XHR object, the technical scope of the AJAX has been expanded in terms of both XML and Asynchronous aspects. Based on these facts, considering the outdated and limited impression of the term "AJAX", it can be replaced by the term "JavaScript-based Simple Pull Delta-Communication" (JS-SPDC) (Dissanayake & Dias, 2017). The term JS-SPCS indicates that it utilizes the SPDC technique, and developed using JS.

III. Technologies and Techniques for the Delta-Communication

After the introduction of AJAX/XHR, the concept of the DC had been used in some other TTs, and each TT is associated with a set of pros and cons. These DC TTs can be mainly classified under data-pull and data-push modes, where the data-pull is based on the request-response model, which the client requests and pull the data from the server; and in the data-push, the server sends data to the client without a request. This section provides an overview of the available DC TTs, indicating their main features. In-depth discussions of the specifications of these TTs are intentionally kept out of the scope of this paper. Instead, the focus is to classify them for a better understanding of their usage, to be utilized in the comparison given in the next section.

A. *AJAX/JS-SPDC*

As discussed before, the AJAX can be seen as the simplest implementation of the DC. It is based on SPDC, works in data-pull mode, and the complexity is comparatively lower than the other DC TTs. The main limitation of the SPDC technique is that it does not support data-push mode; therefore, it is not suitable for real-time data communication as in publisher-subscriber model or any other data-push models.

Several techniques had been introduced to simulate data-push using SPDC; some of them – like polling and long-polling – use the same XHR object, thus also called reverse-AJAX.

B. *Polling (Carbou, 2011)*

In polling, data-push is simulated by sending automatic XHR requests to the server periodically, receiving the response for the frequent requests, and updating the GUI, without the users' explicit requests. There is an overhead of developing periodical requests sending and responses handling. If the frequency of the requesting is low, then the server updates will not be received by the client in real time. To get the updates in real time the frequency of the automatic request sending needs to be higher, but then the network overhead will also be higher. In the cases of where the requests are returned without updates, the resources for processing them are wasted. Addressing these weaknesses of polling, the Comet techniques had been introduced.

C. *Comet (Carbou, 2011)*

Comet is an umbrella term, which covers Streaming and Long Polling. In Comet techniques, unlike polling, the request is held by the server till there are updates to be sent back to the client. In the case of timeouts, the request is terminated, and the client can send a fresh request. This technique reduces the frequency of the requests, thus, also reduces the overhead on the network compared to polling.

1) Streaming: Under streaming, there are two implementation techniques: Hidden iFrame, and Multi-part XHR. In former technique, JS scripts are pushed to the client, and in later technique, a multi-part response is written via the same connection, which the request was sent.

2) Long Polling: Long polling uses the pure SPDC technique as in AJAX and polling, which holds the request for longer time. Compared to streaming, this technique can be seen as evolved and effective.

D. *Server Sent Events (Hickson, 2015)*

On 2015, a true data-push protocol named Server Sent Events (SSE) was introduced, which is unidirectional, from the server to the client. However, it did not become much popular. Compared to data-push simulation techniques, the complexity is lower, and the development and modifiability are easier since the development overhead for data-push simulation as in polling or long polling is not needed.

E. *WebSocket (Fette, 2011)*

An advanced bi-directional DC protocol named WebSocket (WS) was introduced in 2011, which supports both data-pull and data-push modes. WS helps to reduce the number of request-response pairs in the network compared to polling, and the header size of the WS is smaller than HTTP, which leads to increase the scalability by addressing the C10K problem (Kegel, 2014) compared to the other DC TTs. The WS gained the attraction of the web engineers, however, the complexity of WS applications is higher.

IV. Comparison of Some Properties of DC TTs

Table 1 contains the analysis of the contextualized comparison of some selected properties of the DC development TTs. The meaning of the symbols used to denote the values are as follows. Note that these values are comparative to each other, within the context.

- (++) – Higher
- (+) – High
- (-) – Less
- (--) – Lesser

The factor "Core" indicates the abstract technique or the technology used in the core of the technique or the technology. The mode explains whether its data-pull, push, or simulating either push or pull. The data-push simulating TTs (which are actually working on data-pull mode) use the SPDC as their core technique, where SSE and WS use their own protocols.

The complexity is determined by the comparative complexity of the DC implementation, considering the number of components needed for the development of the communication components. SPDC is the simplest form of DC, hence can be considered as the least complex technique. As the extended versions of the SPDC, the TTs such as Polling and Long-polling can be considered as much complex. Though the SSE is a data-push technology, since it is unidirectional, the complexity can be considered as similar to SPDC.

Table 1. Comparison of some features of DC TTs

	JS-SPDC	Polling	Streaming	Long Polling	SSE	WS
Core	SPDC	SPDC	SPDC / hidden iFrame	SPDC	SSE protocol	WS protocol
Mode	Pull	Push simulation	Push simulation	Push simulation	Push	Pull and Push
Complexity	--	+	+	+	-	++
Easiness of Development	++	-	-	-	+	--
Easiness of Maintenance / Modifiability	++	-	-	-	+	--
Scalability	+	--	-	+	+	++

However, the SSE does not need additional components for simulating like in polling or comet, thus can be seen as less complex compared to them. As a bidirectional technology, the complexity of the WS can be seen as the highest. Additionally, WS based development requires additional code and components for WS server implementation and event handling, which makes the development is much complex.

The complexity directly affects the easiness of developing the DC, and also the maintenance and modifiability properties of the system. As the simplest DC technique, SPDC provides the easiest development experience, compared to the other TTs, where WS offers the least easiness as the much complex bi-directional DC technology. Note that this is the comparative easiness of developing a single DC feature, not the actual easiness of the development of the entire system. The comparative easiness of the maintenance and modifiability properties could be a similar to the easiness of the initial development.

For the scalability property, the scalability of the system towards real-time communication based on publisher-subscriber or similar model is considered. For such scenarios, WS offers the capacity to provide the best scalable solution, addressing the C10k issue (Kegel, 2014). JS-SPDC has no mechanism for higher scalable development; however, note that if real-time updates are not needed and if the frequency of the network utilization is less, JS-SPDC can also provide higher scalability. With the frequent redundant requests and responses, polling shows the lowest scalability. Long polling and SSE provides moderate scalability, and considering the higher size of the data communicated the scalability of streaming can be considered lower that long polling and SSE.

V. Facts to Consider in Decision Making of Selecting TTs for RIAs Development

Several important facts to deliberate when making decisions about selecting TTs for the implementation of DC are discussed below (Domenig, n.d.).

Stakeholders' constraints: Stakeholders of the system may have technological constraints such as operating systems, servers, database management systems, etc., based on their available infrastructures and resources. The selection of the DC development TTs may need to align to such constraints since most DC developments are based on frameworks/libraries/plug-ins. For example, if there is a requirement for WS and the stakeholder needs a JAVA based system, a suitable JAVA based framework for WS development should be considered.

Compatibility: The compatibility of the TTs selected for all the client-components, server-components, and DC components is really important. Selection of a TT for one component may affect the TTs of other components, therefore may introduce additional learning curves, and also may introduce limitations for selection of TTs for other components. For example, if WS is selected for DC, for both server and client components, additional frameworks/libraries might be needed, and they should be compatible with each other.

Scalability: The size of the target user population and number of concurrent users (approximate) are the facts to consider when determining the scalability. Scalability plays a significant role when selecting TTs for client-components and DC. If a higher scalability is needed, it is recommended to select WS, regardless its complexity.

Real-time updates: If real-time updates are needed, it is wise to select the DC TT by also considering the scalability. However, to get the true power of the real-time data communication, WS can be seen as the best option. Furthermore, suitable data formats like JSON should be considered to get the maximum support for real-time critical systems (Dissanayake, et al., 2015).

Development, maintenance, and modifiability: The easiness of not only in the initial development, but also in maintenance and future modifications may also take into account. Table 1 contains the easiness in development, maintenance, and modifiability. Correct identification of the scalability requirements is essential, before considering the maintenance and modification aspects.

VI. Conclusion and Future Work

The paper has given an overview of the contemporary DC development TTs, and then has done a contextualized comparison aligning to the facts: communication mode, complexity, easiness of development, easiness of maintenance and modifiability, and scalability. It was identified that the AJAX/JS-SPDC is the simplest implementation of the DC, which incorporates the lowest complexity; however, it lacks in scalability. JS-SPDC is limited to data-pull, thus it does not support data-push. The WS covers the limitations of the other DC TTs like scalability, support for both data pull and push. However, the complexity of WS is seen as the highest among DC development TTs.

Aligning to this comparison, the paper has discussed some facts to be considered in the decision making of selecting proper DC development TTs into RIA development: stakeholders' constraints, compatibility, scalability, need for real-time updates, and the easiness in development/maintenance/modifiability.

In future, we expect to exploit the knowledge delivered in this paper to introduce a taxonomy for the DC development TTs for RIAs. Then we hope to extend that taxonomy to introduce a complete set of taxonomies for RIA development TTs aligning to the architectural elements: Components and Connectors.

References

Carbou, M., 2011. *Reverse Ajax, Part 1: Introduction to Comet,* s.l.: IBM.

Dissanayake, N. R., De Silva, D. & Dias, K., 2015. *A Comparison of the Efficiency of Using HTML over XML and JSON for the Asynchronous Communication in Rich Internet Applications.* Rathmalana, Sri Lanka, s.n.

Dissanayake, N. R. & Dias, K., 2017. Delta Communication: The Power of the Rich Internet Applications [Accepted]. *International Journal of Future Computer and Communication (IJFCC).*

Domenig, M., n.d. *Rich Internet Applications and AJAX - Selecting the best product.* [Online]
Available at: http://www.javalobby.org/articles/ajax-ria-overview/
[Accessed 02 June 2014].

Dutta, S., 2006. *Native XMLHTTPRequest object.* [Online]
Available at: http://blogs.msdn.com/b/ie/archive/2006/01/23/516393.aspx
[Accessed 02 11 2015].

Fette, I., 2011. *The WebSocket Protocol,* s.l.: Internet Engineering Task Force.

Garrett, J. J., 2005. *Ajax: A New Approach to Web Applications.* [Online]
Available at: http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications

Hickson, I., 2015. *Server-Sent Events.* [Online]
Available at: http://www.w3.org/TR/eventsource/
[Accessed 15 May 2015].

Hopmann, A., n.d. *The story of XMLHTTP.* [Online]
Available at: http://www.alexhopmann.com/xmlhttp.htm
[Accessed 02 11 2015].

Kegel, D., 2014. *The C10K problem.* [Online]
Available at: http://www.kegel.com/c10k.html
[Accessed 20 04 2015].

Lawton, G., 2008. New Ways to Build Rich Internet Applications. *Computer,* August, 41(8), pp. 10 - 12.

Salva, S. & Laurencot, P., 2009. *Automatic Ajax application testing.* Venice, s.n., pp. 229 - 234.

Smith, K., 2006. Simplifying Ajax-Style Web Development. *Computer*, May, pp. 98-101.

W3C, 2006. *The XMLHttpRequest Object.* [Online]
Available at: http://www.w3org/TR/2006/WD-XMLHttpRequest-20060405
[Accessed 10 05 2015].