RIA-Bus: A conceptual technique to facilitate the AJAX-based rich Internet application development

Nalaka R. Dissanayake, G. K. A. Dias
University of Colombo School of Computing, Colombo 7, Sri Lanka
and
C. Ranasinghe
Ideahub, Colombo, Sri Lanka

Introduction

Rich Internet Applications (RIAs) have become quite popular in this era of Web2 (Lawton, 2008). Compared to the traditional Web Applications, RIAs are faster, responsive and they have rich Graphical User Interfaces (GUIs) (Paulson, 2005). Among plugin-based approach and script-based approach, Asynchronous Javascript And Xml (AJAX) – which is a script-based technique to develop RIAs – grabbed the demand of the engineers due to many good features of AJAX. AJAX uses stable languages HTML, CSS, JavaScript; it is free; and do not need third party plugins (Farrell & Nezlek, 2007).

However the complexity of the AJAX based RIAs are considered high due to various reasons (Li & Peng, 2012) (Mesbah & Deursen, 2007). Hence, the application of Rapid Application Development (RAD) practices are not supported adequate in AJAX based RIAs engineering (Dissanayake & Dias, The Significance of Importance of an Architectural Pattern for AJAX Based Rich Internet Applications, 2014). To maintain the sustainability of both the developed RIA and the engineering project – not only in the initial development life cycle, but also in post deployment stages – it is important to identify the main cause of these complexities and address it.

Methodology

We conducted a literature survey to gain the background knowledge of the RIAs, AJAX, RAD, architectural patterns and the relationship between these areas. The main intention was to identify and understand the complexities and difficulties of enabling RAD in AJAX based RIA engineering.

As we gain the knowledge and identify some facts related to the complexities in AJAX based RIA engineering, we conducted a cross-sectional survey to verify the facts identified in literature survey. Also understanding of the up to date nature of the AJAX based RIA development was gained by analyzing the data gathered in the cross-sectional survey. The targeted crowed was the individuals engaged in AJAX based RIA engineering; specifically in the design and development phases. Data were gathered using a structured questionnaire with closed end questions and analyzed using statistical methods.

Parallel to the surveys, a series of experiments were conducted to understand and get the experience of AJAX based RIA designing and development techniques, tools, and complexities. This series of experiments was conducted as a prototype based incremental development. In each and every incremental complexities were identified, and some techniques to address the identified complexities were tested. The knowledge gained in the early iterations was applied to the later iterations. We used HTML5 and CSS3 for GUI development, JavaScript for client-side development along with jQuery library, and PHP as the server-side development language. Apache server was used to host the web application and for databases MySQL serer was used.

Results and Discussion

The key finding of the literature survey is, that the main reason for the complexities engaged in AJAX based RIAs engineering is, the lack of availability of architectural formalism for AJAX based RIAs engineering (Dissanayake, Dias, & Jayawardena, 2013).

In the analysis of the cross-sectional survey, we derived and highlighted the following results. The understanding of the general AJAX architecture is not difficult, and there is a good usage of Computer Aided Software Engineering (CASE)

tools in AJAX RIA development. However, the difficulty level of implementing AJAX features in the same page increases, with the number of AJAX features in the page (Dissanayake & Dias, 2014).

One of the complexities we noted throughout the series of experiments is that the difficulties engaged in the file management of the project. Referring to the AJAX general architecture, standard practice when implementing an AJAX feature is, pointing the AJAX request to a dedicated PHP script file in the server. This dedicated script file contains the code for the logic, to provide the complete respond for the particular request. Usually these script files contain few lines of codes, needed to handle the single request. When the number of AJAX requests increases, the number of the script files also increase parallel. To provide more rich features in the application, the need for more AJAX features grows, and as more AJAX features implemented, probably the AJAX request handling script files will be piled up in the server. As the number of script files increases, the logic could be scattered and it may create a complex environment, where the management of the server-side implementation is difficult. This setting will lower the realization of the architectural structure of the system too. Furthermore – due to the aforementioned facts – in the needs of modifications of the RIA, the effort needed will be higher, and this environment may affect the sustainability of the project too.

To control this situation we propose a conceptual technique, we name it as the RIA-Bus (Dissanayake & Dias, 2014). Instead of maintaining numerous small script files in the server to handle the AJAX requests, we advise to keep a single script file – what we call the RIA-Bus – which contains the code to receive all the AJAX requests; direct the processing of the request to the necessary scripts or functions; and respond back the client, with the results returned from the functions in the other script files. Figure 01 illustrates the architectural structure of the RIA-Bus.

A parameter will be sent to the RIA-Bus along with the request by the AJAX engine, to indicate the type of the AJAX feature, which requests the service. Based on the type of the feature, the RIA-Bus can decide, to where the processing should be passed. RIA-Bus is responsible for reading all the data sent by the AJAX engine – using either GET or POST method – and pass the data to the function dedicated for processing the data. Figure 02 shows a sample PHP code of a RIA-Bus.

The script files may contain code for all the logic of the application and organized in a way easy to manage. For an example all the user related functions may be written in one file and products related function may be written in another file. These files can be included into the RIA-Bus as needed and the functions in them can be called according to the type of the feature of the AJAX request.

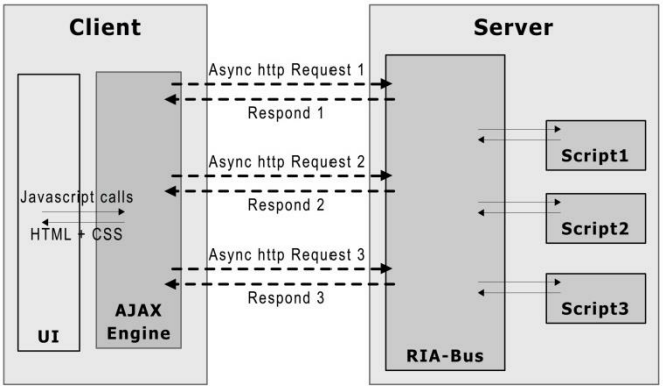

Figure 01: Architecture of AJAX-based RIA, using the RIA-Bus

```
$param = "";
if(isset($_GET["param"]))
{
        $param = $_GET["param"];
}
if(isset($_POST["param"]))
{
        $param = $_POST["param"];
}
if($param=="save")
{
        include_once("userData.php");
        echo saveData($_POST["uName"],$_POST["uAddress"]);
}
if($param=="read")
{
        include_once("userData.php");
        echo readData();
}
```

Figure 02: Sample PHP code of a RIA-Bus

Conclusions

According to the analysis of both literature and cross-sectional surveys we noted that a good understanding of the AJAX general architecture or a higher usage of CASE tools are not capable of reducing the difficulties in implementing multiple AJAX features in the RIA. AJAX based RIAs need good standards and proper architectural formalism (Dissanayake & Dias, 2014).

The integration of the RIA-Bus feature as an architectural concept, will facilitate reducing the number of script files in the server, and organizing the logic in the application in a more structured way. These facilities will provide a better management over the files and the logic, hence will enhance the realization of the RIA and help to maintain the sustainability of the project in both initial engineering and post deployment stages.

In future, we expect to use this RIA-Bus concept to enable better usage of Object Oriented Programming concepts with PHP, in AJAX based RIA development. We plan to incorporate Model-View-Controller architectural pattern with the RIA-Bus, to increase the realization of the RIA, hence reduce the complexities and difficulties in AJAX based RIA engineering (Dissanayake & Dias, 2014).

References

Dissanayake, N. R., & Dias, G. K. (2014). Best Practices for Rapid Application Development of AJAX based Rich Internet Applications. *14th International Conference on Advance in ICT for Emerging Regions*, (pp. 63-66). COlombo.

Dissanayake, N. R., & Dias, G. K. (2014). Essential Features a General AJAX Rich Internet Application Architecture Should Have in Order to Support Rapid Application Development. *International Journal of Future Computer and Communication, 3*(5), 350-353.

Dissanayake, N. R., & Dias, G. K. (2014). The Significance of Importance of an Architectural Pattern for AJAX Based Rich Internet Applications. *KDU International Reseach Conference 2014.* Colombo.

Dissanayake, N. R., & Dias, G. K. (2014). What does the AJAX Rich Internet Applications need to support the Rapid Application Development. *3rd International Conference on Human Computing, Education and Information Management System (ICHCEIMS 2014)*, *15*, pp. 1-4. Sydney, Australia.

Dissanayake, N. R., Dias, G. K., & Jayawardena, M. (2013). An Analysis of Rapid Application Development of AJAX based Rich Internet Applications. *International Conference on Advances in ICT for Emergin Regions (ICTer)*, (p. 284). Colombo, Sri Lanka.

Farrell, J., & Nezlek, G. S. (2007). Rich Internet Applications The Next Stage of Application Development. *Proceedings of the ITI 2007 29th Int. Conf. on Information Technology Interfaces*, (pp. 413 - 418). Cavtat, Croatia.

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

Li, J., & Peng, C. (2012). jQuery-based Ajax General Interactive Architecture. *Software Engineering and Service Science (ICSESS), 2012 IEEE 3rd International Conference*, (pp. 304-306). Beijing.

Mesbah, A., & Deursen, A. v. (2007). An Architectural Style for AJAX. *Software Architecture, 2007. WICSA '07. The Working IEEE/IFIP Conference.* Mumbai.

Paulson, L. D. (2005, Oct). Building rich web applications with Ajax. *Computer, 38*(10), 14 - 17.