Integrating REST with RIA-Bus for Efficient Communication and Modularity in Rich Internet Applications

NR Dissanayake[1#], T Wirasingha[2] and GKA Dias[2]
[1]*University of Colombo School of Computing, Colombo 7, Sri Lanka*
[2]*Informatics Institute of Technology, Wellawatta, Sri Lanka*
[#]*nalakadmnr@gmail.com*

Abstract *— Rich Internet Applications give a rich set of features and an enhanced user experience, but the engineering of Rich Internet Applications comes with complexities, mainly due to poor realization caused by the lack of architectural formalism. If we can realize the architectural properties of Rich Internet Applications well, it might reduce the engineering complexities and the development of Rich Internet Applications may too offer a good experience. In this paper we discuss enhancing the structure and modularity of our proposed concept RIA-Bus, by integrating the customized version of the REST style. We expect to evolve this hybrid style further to come up with an abstract architectural style for Rich Internet Applications.*

Keywords *— Rich Internet Applications, REST, RIA-Bus*

I. INTRODUCTION

Rich Internet Applications (RIAs) have become popular among users; and have gained the attention of the Web2 engineers over last decade (Lawton, 2008). The rich User Interfaces (UI) and the faster responding nature can be seen as main factors for the attraction for RIAs (Piero, et al., 2010). The asynchronous communication mechanism is the key, which enables the development of the rich features in RIAs (Busch & Koch, 2009). There are multiple approaches to implement the asynchronous communication and the script-based approach has become the main choice with its open-source, free and plug-in-less features (Farrell & Nezlek, 2007). The techniques and technologies used for script-based approach have been evolved from the Asynchronous JavaScript and Xml (AJAX) (Garrett, 2005) to Websockets (Fette, 2011) throughout the last decade. However – despite the popularity – the engineering of RIAs is still suffering from complexities (Piero, et al., 2010) (Li & Peng, 2012), hence the designing and developing the RIAs are still complicated tasks (Preciado, et al., 2007).

REpresentational State Transfer (REST) architectural style (Fielding, 2000) has become the de facto standard for the Web2 applications, especially for designing and developing the web services. REST can be used to maintain the modularity of the applications – mainly for the pattern based on Create, Read, Update, Delete (CRUD) operations – applying its simple principle to the Unified Resource Locator (URL) using the GET, POST, PUT and DELETE form methods.

In our ongoing research we try to identify the architectural complexities in RIAs and design an architectural style for RIAs. We have introduced a concept – in our ongoing research – named RIA-Bus (Dissanayake, et al., 2015), to minimize the complexities engaged in the asynchronous communication in AJAX based RIAs. RIA-Bus helps to centralize the asynchronous request handling in the server, thus increases the simplicity and modifiability of the RIA.

In this paper we propose and discuss how the REST can be integrated in to the RIA-Bus, to increase the modularity of the server-side code. We expect that this integration will provide adequate support for various architectural and non-functional quality attributes, such as evolvability, extensibility and customizability.

II. METHODOLOGY

A literature survey was conducted to gain the domain knowledge of RIAs, web architectures, AJAX and other asynchronous communication techniques and technologies.

A cross-sectional survey was conducted to understand the current state of the facts identified in the literature survey. Targeted population was the individuals engaged in RIA development; the data were gathered using a structured questionnaire with closed end questions; and the gathered data were analysed using statistical methods, to derive knowledge.

Parallel to the surveys, we did run a series of experiments to realize the RIA development and gain some empirical

evidence within the domain. The experiments were prototype based and conducted in an incremental development. Issues were identified in each and every iteration and some solutions were introduced. Findings in early iterations were tested, refined and used in later iterations. The empirical evidence was utilized in deriving an architectural style for RIAs. For the client-side development JavaScript (JS) and jQuery; for the server-side development PHP; for the database development MySQL; and AJAX for the asynchronous communication technique were used. Locally installed XAMPP tool was used for Apache web server and MySQL database server.

III. DISCUSSION

In the literature survey we identified some facts, which affect the complexity of the RIA engineering; and we noted that the lack of architectural formalism for RIAs as the major fact (Mesbah & Deursen, 2007). Lack of Model-View-Controller (MVC) separation (Cheung, et al., 2007. December 17-20.); additional learning curves of the new frameworks and libraries; and need for identification of proper tools; are some other facts we identified throughout the literature survey (Dissanayake, et al., 2013).

Analysing the data gathered in the cross-sectional survey, we derived correlation between the number of AJAX features per page and the difficulty level of implementing the AJAX features; such that when the number of AJAX features increases, the level of difficulty increases proportionally (Dissanayake & Dias, 2014).

While conducting the experiments, we identified that the asynchronous communication handling is done in a vague manner, and that leads to have complex/ad-hoc structures of files and coding, which are difficult to maintain. Therefore a concept named RIA-Bus, a server-side component was introduced, which is responsible for handling the asynchronous communication between the client and the server (Dissanayake, et al., 2015). It helps to centralize the asynchronous request handling and the controlling in the server efficiently, hence support the maintenance.

In this paper we discuss the results of the experiments conducted for testing the concept RIA-Bus further. When testing the RIA-Bus in multiple prototypes, we identified some drawbacks as follow.

As the application grows with the number of modules and asynchronous requests, the algorithm in RIA-Bus becomes complex, thus difficult to manage. It may lower the evolvability and the ease of code maintenance of the RIA. Furthermore, we noted that the exposure of the URL of the RIA-Bus – in the client-side code – might introduce some security loop-holes, thus it is not a good practice.

To address these drawbacks we tried incorporating two techniques. The first technique is, enabling the access to the RIA-Bus via a common index file in the server's model directory (Dissanayake & Dias, 2014), without exposing the actual URL of the RIA-Bus in the client-side code. This improves the security, and allows a better management and usage of the RIA-Bus. The second technique is, integrating the REST style to enhance the structural properties of the communication, therefore increase the modularity of the internal algorithm of the RIA-Bus.

The figure 1 illustrates the abstract algorithm of the asynchronous communication processing, utilizing the two proposed techniques.

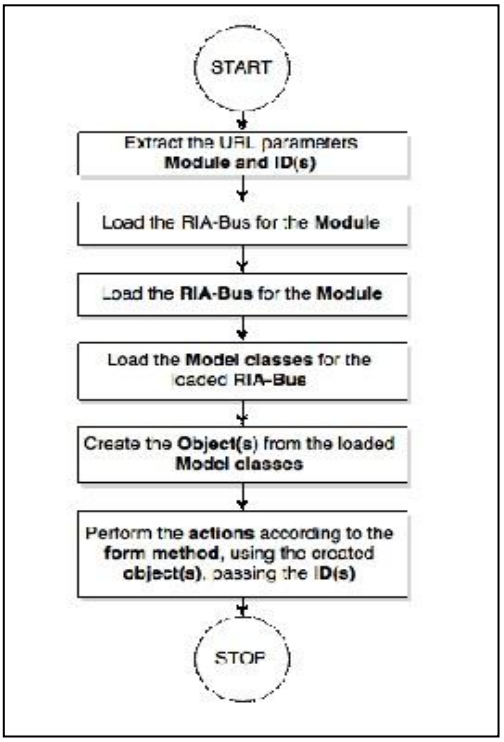

Figure 1: Abstract algorithm of the REST integrated RIA-Bus

A. Common index file

When the RIA-Bus is used, all the AJAX requests are directed to a dedicated file, which is assigned for the RIA-Bus. As the system grows, the developers can maintain separate files for multiple RIA-Buses for different modules. Then the JavaScript in AJAX engine has to use the direct URLs for these dedicated RIA-Bus files. There are two drawbacks here 1) there will be multiple URLs to maintain and 2) the physical addresses of these files are exposed to anyone, which is not a good practice.

```
<modelDirectory>/index.php
```

Figure 2: path to the common index file

To the end of the URL of this common index file in the model directory, we add two additional parameter segments for the AJAX requests, as shown in figure 3. The first segment defines the module and the second segment defines the ID(s) to be processed for the request. If the IDs section is not used, it should be set as *null*.

```
modelDirectory/index.php/User/null

  path to index file   Module ID(s)

modelDirectory/index.php/User/001&002
```

Figure 3: URL to RIA-Bus

Using a small PHP code snippet in the index file as show in figure 4, the data in the parameter segments can be extracted. According to the first URL parameter, the module can be identified and the RIA-Bus for that module can be loaded.

```
<?php

$segments =
    explode('/',
trim(parse_url($_SERVER['REQUEST_URI'],
    PHP_URL_PATH), '/'));

$module = $segments[count($segments) - 2];
$ids = $segments[count($segments) - 1];

if($module == "User")
{
   include("RIABusUsers.php");
}

?>
```

Figure 4: Sample PHP code snippet for the index file in model directory

B. Customized REST version

REST uses four form methods – GET, POST, PUT and DELETE – to define the CRUD pattern. Both POST and PUT are used to submit complete form data other than ID(s). Since we use specific URL segments to add more details to the request, we customized the REST to use only GET, POST and DELETE form methods. We suggest to identify the CRUD based operation, combining the form methods with the ID(s) segment of the proposed URL. The details related to the customized REST style and the CRUD operations are shown in the table 1.

According to the first URL segment, the module is identified – as discussed under the common index file section. The GET form method is used for reading data; POST is used for both inserting and updating entries; and the DELETE is used for removing/deleting entries. Then the availability of the second URL segment is used to add more meaning to the request.

When the second URL segment is null, it's considered as read all; or Insert new entry/entries; or delete all entries. When the ID(s) is/are specified, the request will be processed for the specified ID(s). If there are multiple IDs in the second URL segment, they can be separated using the "&" character, like used in the query string. When the second URL parameter is not null, it is considered as reading or updating or deleting the entry/entries specified by the ID(s). This customized version of the REST style provides more abstract semantic for the RIA-Bus and preserves the consistency of the pattern in the combination of the form methods and the information segments in the proposed URL.

Table 1: Customized version of REST

Form method	URL	Operation
GET	testModel/index.php/User/null	Read all users
GET	testModel/index.php/User/1&2	Read/search specific user(s) [Ids 1 and 2]
POST	testModel/index.php/User/null	Create new user
POST	testModel/index.php/User/1&2	Update specific user(s) [id 1 and 2]
DELETE	testModel/index.php/User/null	Delete all users
DELETE	testModel/index.php/User/1&2	Delete specific user(s) [id 1 and 2]

Figure 5 contains a sample PHP code for the RIA-Bus for the module *User*. In this RIA-Bus, only the classes necessary for the module *User* are loaded. According to the form method and the availability of the ID(s), the CRUD operation is determined as in table 1. Form data need to be utilized when the form method is POST only. Then the POST super global variable is used to read the data from the request, and they can be directly passed as arguments to the parameters of the methods of the module class(es).

```
<?php

include("ModelUsers.php");
$userObj = new Users();

if($_SERVER['REQUEST_METHOD']=="POST"
        && ids=="null")
{
  $userObj->insertUser($_POST["name"]);
}

if($_SERVER['REQUEST_METHOD']=="DELETE")
{
  $userObj->deleteUser(ids);
}
?>
```

Figure 5: Sample PHP code for the RIA-Bus of the module User

IV. CONCLUSION AND FUTURE WORK

RIAs need a more abstract and standard architectural formalism, which describes its characteristics clearly – which are mainly related to the asynchronous communication – to overcome the complexities engaged in the engineering of the RIAs. Asynchronous communication processing in the server-side is a critical area, which creates a "difficult to manage" sections in RIAs. We have proposed a concept to standardize the asynchronous communication processing, using a centralized component named RIA-Bus. The RIA-Bus lacks in modifiability related requirements – mainly the evolvability. When the RIA evolves with modules and features, the RIA-Bus also grows and the maintainability may decrease. Hence we integrate a customized version on REST style with a common index file in the model directory, to have a more controlled development of the asynchronous request handling, and enhance the support for the architectural related non-functional quality attributes such as evolvability, extensibility and customizability.

This is still an ongoing research, and in future we expect to add more constraints into this REST integrated RIA-Bus model and design more abstract and complete architectural model for the RIAs, to reduce the complexities by increasing the realization.

REFERENCES

Busch, M. & Koch, N., 2009. *Rich Internet Applications - State-of-the-Art,* Munchen: Ludwig-Maximilians-Universitat.

Cheung, D. W., Lee, T. Y. & Yee, P. K., 2007. December 17-20.. *Webformer A Rapid Application Development Toolkit for Writing Ajax Web Form Applications.* Bangalore, India, s.n., pp. 248-253.

Dissanayake, N. R. & Dias, G. K. A., 2014. *Best Practices for Rapid Application Development of AJAX based Rich Internet Applications.* COlombo, s.n., pp. 63-66.

Dissanayake, N. R. & Dias, G. K. A., 2014. *What does the AJAX Rich Internet Applications need to support the Rapid Application Development.* Sydney, Australia, s.n., pp. 1-4.

Dissanayake, N. R., Dias, G. K. A. & Jayawardena, M., 2013. *An Analysis of Rapid Application Development of AJAX based Rich Internet Applications.* Colombo, Sri Lanka, s.n., p. 284.

Dissanayake, N. R., Dias, G. K. A. & Ranasinghe, C., 2015. *RIA-Bus: A conceptual technique to facilitate the AJAX-based rich internet application development.* Badulla, Sri Lanka, s.n.

Farrell, J. & Nezlek, G. S., 2007. *Rich Internet Applications The Next Stage of Application Development.* Cavtat, Croatia, s.n., pp. 413 - 418.

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

Fielding, R. T., 2000. *Architectural Styles and the Design of Network-based Software Architectures,* Irvine: University of California.

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

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

Li, J. & Peng, C., 2012. *jQuery-based Ajax General Interactive Architecture.* Beijing, s.n., pp. 304-306.

Mesbah, A. & Deursen, A. v., 2007. *An Architectural Style for AJAX.* Mumbai, s.n.

Piero, F., Gustavo, R. & Fernando, S.-F., 2010. Rich Internet Applications. *Internet Computing, IEEE,* 14(3), pp. 9-12.

Preciado, J. et al., 2007. *Designing Rich Internet Applications with Web Engineering Methodologies.* Paris, IEEE, pp. 23-30.