Abstract concepts: A contemporary requirement for Rich Internet Applications engineering

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

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

[#]nalakadmnr@gmail.com

Abstract— *Rich Internet Applications are very advanced and complex systems, and for their development there are numerous tools, frameworks, libraries, techniques, and technologies are available. The underplaying concepts of the Rich Internet Applications are still have not been defined well, and the tools, frameworks, or libraries do not improve these underlying concepts; instead they might use their own forms of the concepts. If we can understand the abstract fundamental concepts of Rich Internet Applications, we can gain some advantages like: increased realization, knowledge sharing, and lower learning curves. These aspects have not being much discussed or researched within the domain; therefore, we attempt to pinpoint the importance of having abstract concepts for Rich Internet Applications engineering, as a contemporary requirement. This knowledge will help to look at the researching in Rich Internet Application engineering in a different perspective, and will lead to introduce abstract concepts, for Rich Internet Applications.*

Keywords— **Abstract Concepts, Rich Internet Applications, Techniques and Technologies Independency**

I. INTRODUCTION

This section provides the background of the Rich Internet Applications (RIAs) and their development environment, in order to provide a clear picture of the contemporary RIA development setting. Then the problem of the current RIA development environment and what is missing in this setting – indicating the importance of filling this space – are discussed. Finally this section specifies the methodology we used to gather the facts, we focus and discuss in this paper.

A. Background.

RIAs provide enhanced user experience compared to the traditional Web applications, using advanced Graphical user Interfaces (GUIs) and faster rich communication model – commonly known as the "Asynchronous Communication" (Busch & Koch, 2009) (Lawton, 2008) (Koch, et al., 2009). RIAs eliminate page sequence model of the traditional Web applications, thus the work-wait pattern (Manu T S, 2011) (Preciado, et al., 2005), therefore deliver desktop application like user experience, utilizing the advanced Web Techniques and Technologies (TTs). While delivering these features, RIAs have grown into very complex systems, with various components like asynchronous communication engines, which are integrated in diverse ways.

After the introduction of the first open source JavaScript (JS) RIA development technique named Asynchronous Javascript And Xml (AJAX) in 2005 (Garrett, 2005), numerous TTs had been introduced for the engineering of RIAs over the last decade. Techniques like polling and long-polling (Carbou, 2011), which are based on AJAX; and technologies like Server Sent Events (SSE) (Hickson, 2015) and WebSocket (WS) (Fette, 2011) can be given as good examples of TTs, dedicated for the RIA development. Various tools, libraries, and frameworks had also being introduced for RIA engineering rapidly.

B. The problem in the current RIA engineering environment and the motivation

RIAs are very complex systems, with diverse types of components executed in both client and server sides, and these components can be integrated in diverse ways too (Toffetti, et al., 2011) (Dissanayake & Dias, 2014) (Mesbah & Deursen, 2007) (Salva & Laurencot, 2009) (Paulson, 2005) (Prothero, 2006) (Cheung, et al., 2007. December 17-20.) (Dissanayake & Dias, 2014). Proper guidelines, established software engineering methodologies and solutions, and concepts like architectural styles or design patterns, are largely lacking for RIAs (Bozzon, et al., 2006). Even though there are numerous tools, libraries, and frameworks for RIA engineering, they do not improve the underlying concepts (Kuuskeri & Mikkonen, 2009); and also, the available TTs do not describe how to overcome the complexity in design and development of RIAs. Furthermore, these tools, libraries, and frameworks are depending on dedicated TTs (Bozzon, et al., 2006), which leads to exhibit disadvantages as discussed later in this paper. Additionally, it was noted that the available conceptual solutions in the domain are also TTs dependent, thus lack in generality and they do not realize the actual abstract formalism of the RIAs.

In this setting, these available artefacts usually come with high learning curves, making them not so easy – and sometimes not also viable – to adopt, due to some constraints like stakeholders' requirements. For an example even if the developer has good understanding of ASP related

RIA technology, if the stakeholders have JAVA servers, it is not feasible to utilize ASP technology. In one hand, the TTs dependencies introduce limitations for the adoption of the artefact in wider range of platforms/environments; and on the other hand, due to the TTs dependencies combined with high learning curves, the knowledge and experience transferring/sharing between multiple technological environments is also not feasible. These facts lead to cause mandatory inclusion of learning curves when incorporating adoptable alternative solutions into the engineering process. These aspects will be discussed in detail in the rest of the paper.

Considering these fact, to overcome these cons, we think that it is a contemporary requirement for RIA engineering is to have evolution of the abstract concepts, such as architectural styles and design patterns, which are independent from TTs.

C. Methodology

We conducted an intensive literature survey to gain the knowledge of the vast aspects of the RIA engineering, including the specifications of the TTs like AJAX and WS. As we gained the knowledge of the TTs, we conducted a series of experiments to examine the utilization of these TTs, and some available frameworks, and libraries like jQuery (Anon., 2015), AngularJS, and Ratchet (Ratchet, 2016). In these experiments, prototypes were developed to gain empirical evidence, and the knowledge derived was utilized in identifying the abstract formalism of the RIAs and their concepts.

Knowledge gained in the early iterations of the experiments were applied into the later iterations, and the identified facts were continuously refined towards identification of the abstract formalism of the RIAs and realization of the importance of abstract concepts and the abstract architectural formalism of RIAs, which is in the direction of our ongoing research: designing an abstract architectural style for RIAs.

D. Arrangement of the paper

Section II defines the terms TTs Independency and Conceptual Abstraction, in the perspective and the context of this paper. Based on these definitions, section III discusses the advantages of the TTs independency, in the context of RIAs.

Section IV provides the possible abstract concepts for the RIAs, and also briefly discusses and reviews the available work under the given concepts. Section V, which is the last, concludes the papers highlighting the importance of the abstract concepts for the RIAs, and specifying the future work of our ongoing research, which consumes the knowledge of this paper.

II. TECHNIQUES AND TECHNOLOGIES DEPENDANCY AND THE CONCEPTUAL ABSTRACTION

In order to discuss the need for the abstract concepts for RIAs and their advantaged, a good understanding about the meaning of the terms “Techniques and Technologies Dependency” and “Abstract Concepts” is needed. This section discusses the notion of these terms, in the perspective and the context of the focus of this paper.

A. Techniques and Technologies Dependency and its limitations

The best way to explain the TTs dependency is through an example. Consider the Oracle’s Model2 architecture (Oracle, n.d.), which is introduced for – and highly depending on – JAVA based web applications development. Since it is based on JAVA, it does not assist in realizing the common characteristics of RIAs, or adopting the same architecture in any other environments like PHP or ASP. When it comes to engineering of PHP based RIAs, a completely different compatible architectural formalism should be utilized, which might introduce additional learning curves; and the knowledge or experience gained from Model2 architecture will not be useful with these alternatives, since they do not share common characteristics. Therefore we can say that the Model2 has a TTs dependency for JAVA.

Likewise, the currently available concepts, tools, and frameworks are depending on specific set of TTs, and additionally the support for the adoption of other available TTs out of the specified set of TTs is low or none. When trying to adopt other TTs, additional learning curves and incompatibilities can be occurred. Furthermore, the engineers of RIA projects will experience similar situations when trying to adopt newly introduced concepts in future, due to the unavailability of general conceptual formalism for RIAs, which share common characteristics. Since the new concepts may come with their own TTs dependencies and constraints, they will incorporate learning curves, even though they all are in the same RIA domain. Therefore the TTs dependency can be seen as a great limitation, barrier, and a problem for RIA engineering.

B. Conceptual Abstraction

The aforementioned problem can be solved by introducing concepts for RIAs, in more abstract form. The term "abstract" is used in the context of this paper to denote the notion of "independent from TTs", which can ensure the ability of adoption in wider environments, incorporating a lower learning curve. Object Oriented Design and Development (OODD) paradigm, and the Client-Server architectural style (CS style) are good examples of TTs independent abstract concepts. Once the concept is realized, it can be adopted in wider environments, and can be applied and utilized regardless the TTs used. Furthermore, the adoption of these concepts into development with any supporting TTs is relatively simpler.

It has been already seen that the researchers of RIAs are overlooking the technological aspects, resulting gaps between the problem concepts that capture their models, and how these concepts are ultimately implemented through components (Meliá, et al., 2010). We believe that the abstract concepts like, architectural styles and design patterns will be helpful to avoid such technological gaps. Therefore our effort of this paper is to highlight the advantages of the conceptual abstraction and discuss the possible abstract concepts for the engineering of RIAs.

III. ADVANTAGES OF THE TTS INDEPENDENCY THROUGH CONCEPTUAL ABSTRACTION

Before discussing the possible types of abstract concepts for RIA and their usage, it's worth looking into the advantageous we can gain from the conceptual abstraction via the TTs independency. In this section we discuss the advantageous of the TTs independent abstract concepts, based on our empirical evidence gained through the experiments, in the context of RIAs.

Conceptual abstraction of common characteristics: For concepts to be TTs independent, they need to understand, incorporate, and exhibit the common abstract characteristics related to the RIAs, which are independent from the TTs specific concepts. Available concepts for RIAs show TTs dependencies, which means that they have not realized the abstract features of the RIAs, thus the abstract architectural formalism of RIAs. This results in lack of understanding of the common characteristics and essential features of the RIAs; and vice versa, may be the available concepts are TTs dependant since they find its complex to identify the common characteristics of the RIAs.

For RIAs, identification of the common characteristics of the client-side components, server-side components, asynchronous communication related components, non-asynchronous communication components, and their configurations is really essential. This knowledge can be utilized to derive the abstraction of the concepts behind these components, in the direction of identifying the architectural formalism of the RIAs. Furthermore, Abstract concepts can assist in increasing the realization of the common characteristics in a system, which leads to deliver the advantages discussed below.

Increased realization: The abstract concepts will deliver strong realization of the common characteristics, thus the architectural elements and their configurations, without depending on any TTs specific features, which leads to derive the abstract architectural formalism of the RIAs. The realization of the abstract architectural formalism helps in reducing the complexities of the RIAs systems.

Knowledge sharing: Since the conceptual abstraction provides increased realization, the knowledge and the experience gained from one environment can be transferred and shared into other environments easily, regardless the TTs used. For an example, the knowledge and the experience of the use of the client-server style in JAVA web applications can be applied into PHP environment with less effort, since the client-server style is an abstract concept. If the abstract architectural formalism of the RIAs have been understood, sharing knowledge and experience across projects regardless the TTs will be easier, and it will be then a matter of adopting TTs into the abstract architectural formalism, instead of trying to adopt the architectural concepts into the target platform, which is relatively costly.

Lower learning curve: Because of the easiness in knowledge sharing through the increased realization provided by the conceptual abstraction of the common characteristics, learning curve of new RIA systems can be reduced, since they all share the same abstract common characteristics and/or the architectural formalism. This will help to reduce the difficulties in design and development of new RIAs, regardless the platform/environment and the TTs used.

Assist in better TTs selection and adoption: The abstraction provided by the TTs independency will provide flexibility in incorporating available TTs; therefore, it will assist in selecting the proper – or may be better – TTs and adopt them easier. In addition to that, it will assist in understanding the adoption of the new TTs introduced in future. Adequate

realization of the abstract architectural elements of RIAs will provide the capacity for better understand of the applicability of the future TTs. Therefore, better decision making can be achieved, when incorporating TTs into the engineering projects, having a low adoption learning curves.

IV. ABSTRACT CONCEPTS FOR RICH INTERNET APPLICATIONS

This section discusses the possible abstract concept to be utilized in design and development of RIAs. This section also briefly discusses and reviews some available work under the given abstract concepts.

A. Algorithms

Algorithms are originated in the domain of mathematics, however now widely used in computer science and software engineering. An algorithms provides a step by step procedure for solving a specific problem (TechTarget, 2014), and usually TTs independent and abstract. Searching and sorting abstract algorithms are widely used in software engineering.

We think that the asynchronous communication might get advantages of abstract algorithms. The AJAX handling in client-side using pure JavaScript (JS) exhibit an abstract concept and it had been utilized in JS libraries like jQuery. Other than that, we haven't come across any abstract algorithms for RIAs. We believe that the asynchronous communication handling in the server-side might also contain some abstract algorithms, especially when the REST (Fielding, 2000) style is used, and currently we are experimenting on that.

B. Design patterns

Design patterns provide reusable micro-architectures that contribute to an overall system architecture (Gamma, et al., 1993), and they help in reducing the complexity and increasing the reusability in system design and development. Currently the design patterns are also used in GUI development (Granlund, et al., 2001). They can be utilized for rich GUI design and development of RIAs (Duyne, et al., 2002), and helpful in delivering rich features of RIAs with less design and development efforts (Thung, et al., 2010).

Some useful available GUI design patterns for RIAs can be seen as the form pattern (Neil, n.d.), pagination pattern (Thung, et al., 2010), and data displaying patterns such as grid/table, list, tree, and accordion (Neil, n.d.). Create, Read, update, Delete (CRUD) pattern (Yoder, et al., n.d.) is also a useful abstract generic design pattern, which can be utilized for the server-side components of RIAs.

C. Architectural styles

Architectural styles offer a framework for designing system architectures, and styles help in capturing the knowledge of successful systems in past (Selfa, et al., 2006). We have come across several architectural styles for RIAs, however they are depending on some TTs, therefore not abstract. jAGA style (Li & Peng, 2012) is based on jQuery, and the SPIAR style (Mesbah & Deursen, 2007) depends on the characteristics of the frameworks Echo2, GWT, and Backbase.

AJAX general architecture (Garrett, 2005), itself can be considered as an architectural style, however it is based on JS. Anyhow we think the abstract concept of the AJAX general architecture can be extracted to make a TTs independent style, to be also used in desktop application components, which communicates with web servers.

We are working on a research towards an abstract architectural style for RIAs, which can realize the architectural formalism of the RIAs, in the direction of reducing the complexity. Current we have introduced some styles for RIAs: the Balanced Client style (Dissanayake, et al., 2015) provides some guidelines for adoption of Model-View-Controller (MVC) pattern into the RIAs; and the RIA-Bus style (Dissanayake, et al., 2015) provides a mechanism for effective rich communication handling, in the server-side of the RIAs.

V. CONCLUSION

We conclude the paper highlighting that the need for abstract concepts is a contemporary requirement in the domain of RIA engineering. The TTs independency of the abstract concepts can provide the discussed advantages, which we think are essential in the current setting.

We expect that the knowledge of the importance of the abstract concepts for the RIAs – as discussed in this paper – will direct the researching in the domain of the RIAs into a new dimension.

Our ongoing project is focusing on designing an abstract architectural style for RIAs, in order to reduce the complexity in engineering RIAs, by increasing the realization of the abstract architectural formalism of them.

REFERENCES

Anon., 2015. *jQuery.* [Online] Available at: https://jquery.com/ [Accessed 15 May 2015].

Bozzon, A., Comai, S., Fraternali, P. & Carughi, G. T., 2006. *Conceptual modeling and code generation for rich internet applications.* New York, s.n., pp. 353 - 360.

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

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

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. 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), pp. 350-353.

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

Dissanayake, N. R., Liyanage, U. & Dias, K., 2015. *A CONCEPT OF BALANCED-CLIENT FOR RICH INTERNET APPLICATIONS.* Malabe, Sri Lanka, s.n.

Duyne, D. K., Landay, J. A. & Hong, J. I., 2002. Making the Most of Web Design Patterns. In: *The Design of Sites.* s.l.:Pearson Education, Inc.

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.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J., 1993. *Design Patterns: Abstraction and Reuse of Object-Oriented Design.* s.l., s.n.

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

Granlund, Å., Lafrenière, D. & Carr, D. A., 2001. *A Pattern-Supported Approach to the User Interface Design Process.* New Orleans, s.n.

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

Koch, N., Pigerl, M., Zhang, G. & Morozova, T., 2009. *Patterns for the Model-Based Development of RIAs.* Heidelberg, s.n.

Kuuskeri, J. & Mikkonen, T., 2009. *Partitioning Web Applications Between the Server and the Client.* Honolulu, Hawaii, U.S.A., s.n.

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.

Manu T S, 2011. *Advances & Applications in Ajax,* s.l.: CUSAT.

Meliá, S., Gómez, J., Pérez, S. & Díaz, O., 2010. Architectural and Technological Variability in Rich Internet Applications. *IEEE INTERNET COMPUTING*, May/June, pp. 24-32.

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

Neil, T., n.d. *Standard Screen Patterns v3.0.* s.l.:s.n.

Oracle, n.d. *About the Model 2 Versus Model 1 Architecture.* [Online] Available at: http://download.oracle.com/otn_hosted_doc/jdeveloper/1012/developing_mvc_applications/adf_aboutmvc2.html [Accessed 15 06 2015].

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

Preciado, J., Linaje, M., Sanchez, F. & Comai, S., 2005. *Necessity of methodologie to model Rich Internet Applications.* s.l., s.n.

Prothero, J., 2006. *Ajax Usability Benefits and Best Practices,* s.l.: JackBe Corporation.

Ratchet, 2016. *Ratchet WebSockets for PHP.* [Online] Available at: http://socketo.me/ [Accessed 10 05 2016].

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

Selfa, D. M., Carrillo, M. & Boone, M. d. R., 2006. *A Database and Web Application Based on MVC Architecture.* s.l., IEEE Computer Society, p. 48.

TechTarget, 2014. *Algorithm.* [Online] Available at: http://whatis.techtarget.com/definition/algorithm [Accessed 20 05 2016].

Thung, P. L., Ng, C. J., Thung, S. J. & Sulaiman, S., 2010. *Improving a Web Application Using Design Patterns.* Kuala Lumpur, s.n.

Toffetti, G., Comai, S., Preciado, J. C. & Linaje, M., 2011. State-of-the Art and trends in the Systematic Development of Rich Internet Applications. *Journal of Web Engineering,* 10(1), pp. 70-86.

Yoder, J. W., Johnson, R. E. & Wilson, Q. D., n.d. *Connecting Business Objects to Relational Databases,* s.l.: s.n.