

# Implementation of Multiplatform RIA using User Interface Components

K.MuthuManickam M.E.,

Senior Lecturer

Department of Computer Science and Engineering

Arunai Engineering College, Tiruvannamalai – 606 603

Tamilnadu, India

**Abstract**—Nowadays, there are a growing number of Web 1.0 applications that are migrating towards Web 2.0 User Interfaces, in search of multimedia support and higher levels of interaction among other features. These Web 2.0 features can be implemented using RIA technologies. However, most of the current Web Models do not fully exploit all the potential benefits of Rich Internet Applications. Although there are interesting works that extend existing methodologies to deal with RIA features, they do not fully exploit presentation issues. RUX is a method that focuses on the enrichment of the User Interface while takes full advantage of the functionality already provided by the existing Web models. Far from explaining RUX-Method in detail, this paper focuses on the way the method deals with the definition and transformation of components at different levels of abstraction for different RIA rendering platforms.

**Keywords:** RIA, UI, AJAX, RUX

## I. INTRODUCTION

With the appearance of Web 2.0, the complexity of tasks performed via Web applications User Interfaces (UIs) has been increasing, in particular when high levels of interaction, client-side processing, and multimedia capacities have to be performed. In this context traditional HTTP-HTML-based Web (Web 1.0) applications are showing their limits, presenting several restrictions. To cite a few, they have *Process limitations* (e.g., complex Web Applications often require that the user navigates through a series of pages to complete only one task); *Data limitations* (e.g., interactive explorations of the data are not allowed); *Configuration limitations* (e.g., many Webs require the configuration of a product/service from multiple choices, but in general, they are unable to present the customized product/service to the user in an intuitive way and in a single step) and *Feedback limitations* (e.g., continued and ordered interaction without page refreshments is not allowed, so the interaction of the user is quite limited). These are some reasons why developers are building the future of the Web using Web 2.0 UIs technologies by means of Rich Internet Applications

(RIAs). RIAs overcome the limits mentioned above, combining the benefits of the Web distribution architecture with the interface interactivity and multimedia support available in desktop applications. Some of the novel features of RIAs are User Interface (UI) and the interaction paradigm; others extend to architectural issues, such as, the client-server communication and the distribution of the data and business logic. They support online and offline usage, sophisticated UIs, data storage and processing capabilities directly at the client side, powerful interaction tools leading to better usability and personalization, lower bandwidth consumption, and better separation between Presentation and content. Although traditional Web methodologies are been extended in several directions to cope with some of these new features, currently they do not cover RIA composition parameters fully at all. Most Web methodologies do not support multimedia properly, their focus being on data intensive Web applications. In addition, most of the multimedia methodologies are not data intensive and business logic oriented because they mainly focus on presentation, temporal specifications to support multimedia/animations and final-user interaction.

We can conclude that there is a need for methods and tools for the systematic development of RIAs, particularly for the Presentation level. In this sense, RUX-Method (Rich User experienced is a model driven method which supports the design of multimedia, multi-modal, multiplatform and multi-device interactive Web 2.0 UIs for RIAs. For this purpose, RUX-Method makes use of a Component Library for the definition of Components for different RIA rendering platforms. This definition is made at different levels of abstraction. This Library also specifies the transformation between Components placed at different Interface levels. The objective of this paper is briefly presenting this Library and the Components from the theory to the practice and the way in which RUX Method deals with the generation of RIA components. The rest of the paper is as follows.

## II. CONCEPTS AND ECHNOLOGIES IN RIAS

After several years in existence the Internet is mature and the primary platform for Graphical User Interfaces. Though it has delivered well, people missed the richness of the old fashioned desktop application. In order to address this issue, very first steps were taken using AJAX. This was the evolution of Rich Internet Application (RIA). Initial adopters solved the problem using the combination of AJAX and DHTML. This

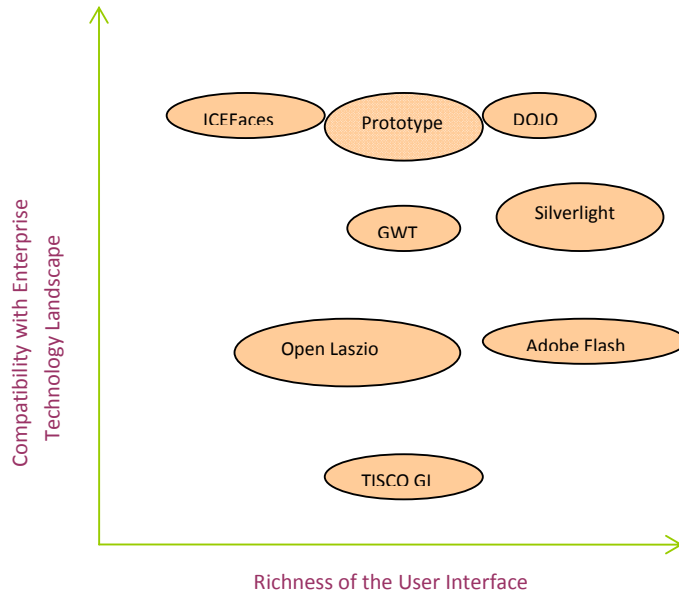


Figure 1. Richness Vs Compatibility

approach gave a major boost to the web application development and became the front runner in the RIA technology. Though this approach brought a lot of richness, it was still not close enough to the richness of the desktop application. So, finally evolved the era of browser plug-ins. Few plug-ins were already in existence, hence adaptation of them was easy. Competition brought more plug-ins to the market. Along came the constraints and complexities involved in the decision making. Some of the prevalent technologies, their features, implementation nuances and the possible future of Rich Internet Applications are shown in Fig.1.

## III. ANATOMY OF RIA

From a high level architecture perspective, the design of a RIA application is almost the same as any web application, except for the fact that the server request doesn't refresh the whole page. Each request is sent in an asynchronous way and handled accordingly by the client and the server. Plug-in based applications generally run within a plug-in runtime. This runtime provides mechanism for the server communication. For

non plug-in based applications, server communication is generally handled by an AJAX library.

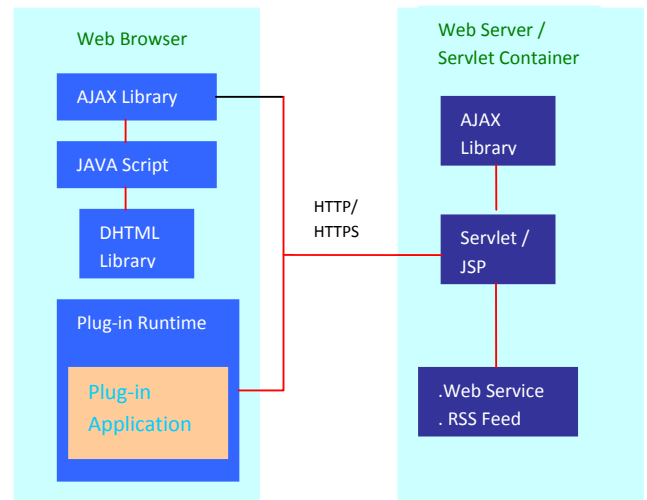


Figure 2. AJAX Request

Prevalent implementations there are many implementations available today in the market. They vary from simple AJAX JavaScript libraries to full fledge AJAX framework and plug-ins. One of such a implementation is shown in the above figure as Ajax Request. If we have to categorize the available products in the market, they can be categorized in three broad categories:

- **AJAX libraries:** These libraries provide graphical (DHTML) components along with JavaScript libraries. While graphical components provide the richness, JavaScript libraries provide the asynchronous communication. Most of these products can easily be integrated with the existing enterprise technology landscape.
- **Plug-ins:** These products provide the best richness in terms of graphical user interface. Product vendors use their proprietary language for development and delivery of content. Though the content is delivered over HTTP, the structure is not understood publicly.
- **Proprietary Tools:** Some of the tools in this category produce results which are publicly understood while others are not. Some use their proprietary language" while others use an industry standard language, like Java. From a development stand point; these products are close to plug-in based products. But compiled end products run seamlessly on a browser. In fact, some give an option to compile in DHTML or plug-in consumable format.

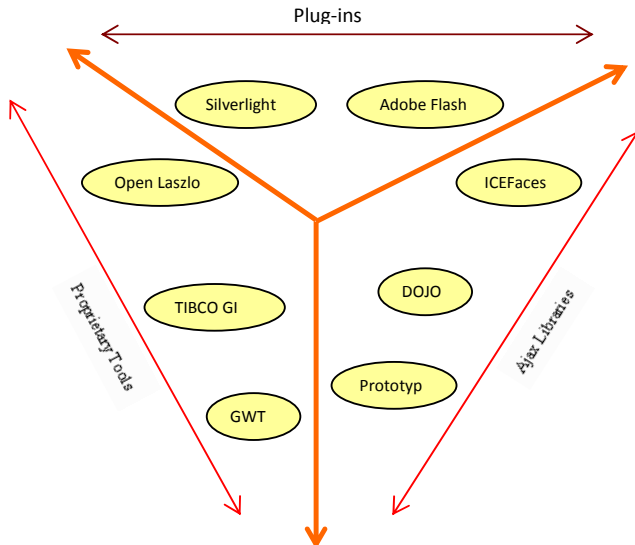


Figure 3. Three Broad Categories of RIA Technology

#### IV. RUX-METHOD IN BRIEF

RUX-Method is a model driven method which Supports the design of multimedia, multi-modal and multiservice interactive UIs for RIAs. RUX-Method focuses on the enrichment of the User Interface while takes full advantage of the content and functionality already provided by the existing Web models. At design time, RUX-Method uses existing data, business logic and presentation models offered by the underlying Web model being enriched. This information provides a UI abstraction which is transformed until the desired RIA UI is reached. At run time, while a new UI is generated from RUX-Method, the data and business logic remain the same. To sum up, the responsibility of RUXMethod is providing a new UI with RIA features.

To facilitate the UI development process, RUX-Method is divided into three Interface levels: Abstract, Concrete and Final Interfaces. Each Interface level is mainly composed by Interface Components whose specifications are stored in the Component Library. One Component can only belong to one Interface level. The Library also stores how the transformations among Components of different levels are carried out. There are two kinds of adaptation phases in the RUXMethod according to the Interface levels defined above. Firstly, the adaptation phase that catches and adapts Web 1.0 (data and navigation, as well as presentation when it is possible) to RUX-Method Abstract Interface that is called Connection Rules (CR). Secondly, the adaptation phase that adapts this Abstract Interface to one or more particular devices and grants access to the business logic that is called Transformation Rules 1 (TR1). Finally, there is an additional transformation phase, Transformation Rules 2, (TR2) that completes the MDA life-cycle of RUX-Method supporting and ensuring the right code generation. Thus, in TR2, the Final Interface is automatically obtained depending on the chosen RIA rendering technology (e.g. Laszlo, AJAX). This process

is performed automatically because TR2 establishes the way the matching takes place among Concrete Interface Components and Final Interface Components. A RUX-Method overview is depicted graphically shown in the below Figure.

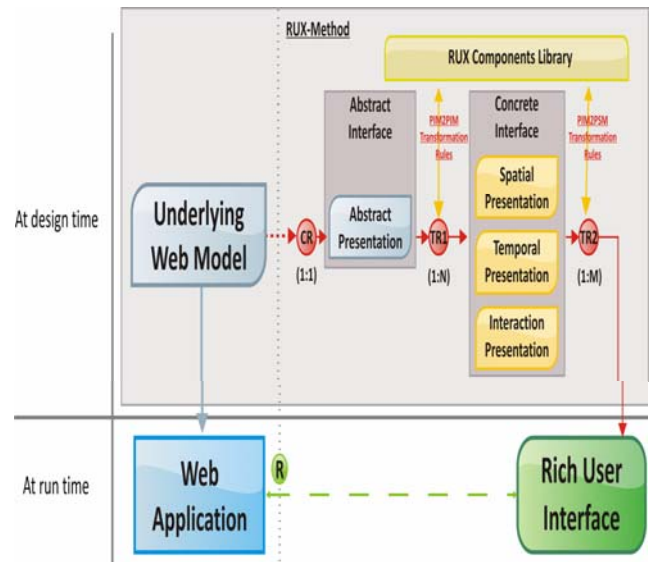


Figure 4. RUX Method

In this document and due to the RUX-Method nature, when we talk about components, we refer to User Interface Components. These Components can have different complexity levels and intrinsic functionality (e.g. Widgets, Gadgets, etc.).

The Components Library is responsible for:

- 1) Storing the component specification (name, methods, properties and events),
- 2) Specifying the transformation/mapping features for each Component from an Interface level into other Component in the following Interface level
- 3) Keeping the hierarchy among Components at each Interface level independently from other levels. The set of Interface Components defined in the Library can be increased or modified by the modeller according to the specifications of the project. The set of available transformations can be also increased or updated according to the Interface Components included in the Library. For a given Component several transformations can be defined depending on the target interface level. The Component Library stores the interface level structure by means of skeletons.

#### V. COMPONENTS AND TRANSFORMATIONS

Components are used for solving specific interface tasks. In the Abstract Interface the different kinds of media and views define the grouping and the type of

elements that the user is going to perceive. These Abstract Interface Components are transformed (by means of TR1) into Concrete Interface Components according to the different ones available in RUX-Method: *Control* Components that are used for data I/O (e.g. *textcontrol*, etc.), *Layout* Components used to organize the content (e.g. *HDivideBox*, etc.) and *Navigational* Components that are used for navigating the content (e.g. *tabnav*, etc). Each Component can be composed by different parts (not all of them required according to the Interface level of the Component): *Name* specifies the Component name and the list of Components from the previous Interface level that can be transformed in this Component. *Capability* expresses the functionality needed to express the Component behaviour. It is composed of a Header and a Body. While the Header is used and shared by all the instances of this Component in the application, the Body is specific for each instance. To clarify this abstract specification, let us show an example for an AJAX specific component: the Header can be a snippet CSS or JavaScript function placed once in the application. On the other hand, the Body is placed once for each instance defining different values in each case as necessary (e.g., example values for size, font color, etc), sharing all the instances of a component the same header. A *Property* indicates a Component characteristic that contains a value according to its Interface level.

A skeleton may include a set of common resources needed by Components. Regarding the transformation among Components, both Transformation Rules (TR1 and TR2) follow the same steps:

- 1) Use the skeleton defined (in the Component Library) for the target interface level,
- 2) Transform each Component in the source interface level into its corresponding default Component in the target interface level,
- 3) Enrich this skeleton including the Components obtained from step 2. This process distributes the Component Headers and Bodies across the skeleton preserving the Component.

## VI. CONCLUSIONS

This paper has shown how RUX-Method allows the definition of Components at different levels of abstraction for the generation of multiplatform RIA User Interfaces. The definition of Components and the specification of transformation among Components are done in the Components Library. With the aid of this Library, it is possible to adapt an old Web 1.0 User Interface obtained following a Web methodology to a new Web 2.0 User Interface with multimedia support, richer user interactions and custom Components able to

be rendered in different platforms. All these specifications are carried out by a module of RUX-Tool named Component Library editor, which makes possible to manage the whole set of Components and the relations among them. Maybe a missing part of this work is related with the evaluation and comparison with other proposals. However, due to the originality of RUX-Method, it is not possible this type of content. Maybe the most relevant proposal regarding RIA UIs. Latter has also tool support, but the proposal miss some RIA UI features like temporal behaviour and do not use a previous Web model to full-fill the model-driven Web application development. While theoretically RUX-Method can be combined with many of the existing Web models.

## REFERENCES

- [1] Ceri S., Fraternali P., Bongio A., Brambill M., Comai S., and Matera M., *Designing Data Intensive Web Applications*, Morgan Kauffmann, San Francisco, 2002.
- [2] Gómez J. and Cachero C., "OO-H Method: extending UML to model web interfaces", *Information modeling for internet applications*, Idea Group Publishing, 2003.
- [3] Koch, N., Knapp, A., Zhang, G., and Baumeister, H., "UML-Based Web Engineering: An Approach Based on Standards", *Web Engineering: Modelling and Implementing Web Applications*, Human-Computer Interaction Series, vol. 12, chapter 7, Springer-Verlag, 2007, pp 157-191.
- [4] Linaje M., Preciado J.C., Sánchez-Figueroa F., "Engineering Rich Internet Application User Interfaces over Legacy Web Models," *IEEE Internet Computing*, vol.11, no.6, pp.53-59, Nov.-Dec, 2007.
- [5] Preciado J.C., Linaje M., Comai S., and Sanchez- Figueroa F., "Designing Rich Internet Applications with Web Engineering Methodologies", *International Symposium on Web Site Evolution*, IEEE, 2007, pp. 23- 30.
- [6] RUXProject Homepage: [www.ruxproject.org](http://www.ruxproject.org).
- [7] Martínez-Ruiz FJ, Muñoz Arteaga J, Vanderdonck J, teal (2006) A first draft of a Model-driven Method for Designing Graphical User Interfaces of Rich Internet Applications. *Proceedings of LA-Web 32-38*.
- [8] Urbieto, M., Rossi, G., Ginzburg, J., Schwabe, D.: *Designing the Interface of Rich Internet Applications*. In *Proc. of the 5th Latin American Web Congress*, pp.144-153, IEEE (2007).