Journal of Universal Computer Science, vol. 14, no. 13 (2008), 2239-2254
submitted: 31/10/07, accepted: 27/6/08, appeared: 1/7/08 © J.UCS

Adapting Web 1.0 User Interfaces to Web 2.0 Multidevice User Interfaces using RUX-Method

Juan Carlos Preciado
(Quercus Software Engineering Group, Universidad de Extremadura
Escuela Politécnica, Cáceres, Spain
jcpreciado@unex.es)

Marino Linaje
(Quercus Software Engineering Group, Universidad de Extremadura
Escuela Politécnica, Cáceres, Spain
mlinaje@unex.es)

Fernando Sanchez-Figueroa
(Quercus Software Engineering Group, Universidad de Extremadura
Escuela Politécnica, Cáceres, Spain
fernando@unex.es)

Abstract: The development of Web applications, both functionality and Web User Interfaces (UIs), has been facilitated over the last few years using Web models and methodologies. However, new requirements that overcome traditional HTML-based Web 1.0 User Interfaces limits have arisen. Developers and tool vendors have answered these limits introducing Rich Internet Applications (RIAs). RIA technologies provide Web 2.0 UI capabilities such as high interactivity and native multimedia support among others. Currently, numerous developers are adapting many of their legacy Web 1.0 applications to Web 2.0 introducing Web 2.0 UI capacities while maintaining the business logic. Nevertheless, there is a lack of methodologies to support this adaptation process. In this paper we show how to use a model driven method called RUX-Method for the systematic adaptation of existing Web 1.0 UIs to Web 2.0 multidevice UIs. This method focuses on new UI capabilities provided by RIAs while taking advantage of functionality already provided by existing Web models. The proposal follows a common UI design for all the devices and an ad-hoc design approach for each device attending to its specific features.

Keywords: Adaptation Techniques, Web Engineering
Categories: H.3.5, H.5.2, H.5.4

1 Introduction

The growth of the Internet is a reality and its impact on society continues to increase in business, education, industry, etc. The Web has become a common platform for providing access to information in many different formats and its distributed architecture has inherited benefits such as low maintenance costs, decentralization and resource sharing among others.

Over the past few years, the expansion of the Internet has been supported by Web 1.0 HTML-based applications and the Web Engineering community has proposed

methodologies to support the design, development, and maintenance of these Web 1.0 applications. The abstract specification of these applications is supported by some methodologies offering Web models (e.g. WebML, OOHDM, UWE, OO-H among others [Preciado, 05]) and some of these are also able to provide automatic code generation offering associated CASE tools.

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 and developers are building the future of the Web using Web 2.0 UI technologies such as Rich Internet Applications (RIAs), also known as Rich Web Applications [Brent, 07]. RIAs combine the benefits of the Web distribution architecture with the interface interactivity and multimedia support available in desktop applications. For this reason, more and more developers are trying to adapt their applications by replacing the old UI with a new one using RIAs.

Several authors have highlighted that the UI development is one of the most resource-consuming stages of application development, especially when Web applications are developed [Daniel, 07]. However, currently there is a lack of models and methodologies to support the RIA UI design [Preciado, 05]. Although there are some research projects trying to incorporate RIA features in their methodologies [Bozzon, 06] [Urbieta, 07] they do not cover the majority of RIA features. Moreover, they do not support the systematic adaptation from Web 1.0 UIs to Web 2.0 UIs.

This provides researchers an open issue because it is not only important to have methodologies to develop Web 2.0 applications from scratch, but it is also important to adapt existing Web 1.0 applications to Web 2.0 following a methodology. For example, YouTube, Amazon and Yahoo have all recently introduced Adobe Flash in the form of RIA widgets with dynamic content into their main web pages to improve user experience. Because of this, there is a real need for systematic methods and tools to perform this necessary adaptation in order to preserve consistency and improve reusability.

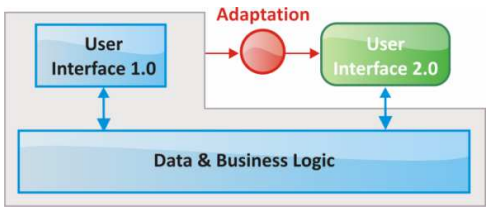

Figure 1: Adaptation of UI: from Web 1.0 to Web 2.0

The contribution of this paper is to present the adaptation (Figure 1) of legacy model-based Web 1.0 applications to multidevice Web 2.0 UIs through RUX-Method (Rich User eXperience Method) [Linaje, 07]. RUX-Method uses existing data and business logic offered by the Web application being adapted, providing a UI abstraction which is then transformed until the desired RIA UI is reached. The proposal follows a common UI design for devices such as PDAs, smartphones, etc. and RIA development platforms such as FLEX, AJAX, etc. In this sense, a change in

the UI requirements of the application must only be done once. Notwithstanding, in order to achieve a better user experience for the different RIA capable devices, RUX-Method supports the UI ad-hoc design for each device according to its features (e.g. screen size or supported rendering technologies). This is mandatory because RUX needs a full understanding of the legacy Web application, and the best way to do this is by means of models.

Following [Canal, 06] the proposed technique is Static Time Adaptation which can be performed both manual and automatic according to the categorization proposed. However, dynamic adaptation is taken into account for adapting the UI in ubiquitous and mobile computing scenarios. According to [Yahiaoui, 04] it is a technical adaptation.

RUX-Method is accompanied by a tool (RUX-Tool) that supports full automatic code generation. The method shown in this document has been validated by several case studies using RUX-Tool. The CASE tool and some video demos are available online in the RUX-Project homepage at *http://www.ruxproject.org/*.

The rest of the paper is as follows. Section 2 presents an overview of Rich Internet Applications by means of an example. Then, in section 3 RUX-Method will be briefly introduced. Section 4 shows the adaptation method in RUX-Tool while section 5 applies this process to the example shown in section 2. Finally, conclusions and future work are shown in section 6.

2 RIA overview and motivating example

Web Interactive Applications [Suh, 05] are defined as Web applications where a client program uses the Web infrastructure in order to reach end-users through Web based GUI (usually requires a plug-in installation on the client-browser). RIAs fit well into this category. While RIAs include functionality inherited from multimedia desktop applications (native multimedia support and so on), they also offer new features that have not been presented together inside Web 1.0 applications before. RIA concepts can be implemented over multiple development platform technologies such as Adobe Flex or OpenLaszlo [Brent, 07].

The concept of richness in RIA extends the four main aspects (*data*, *business logic*, *presentation* and *communication*) of Web 1.0 applications [Preciado, 07]. Business logic design is commonly achieved by Web models through the combination formed by hypertext, navigation and process design. These four extensions involve both, client and server sides. In order to understand the need for the adaptation process, next we show these extensions through a simple example. The selected example is a hotel booking Web application. First we will show the problems of Web 1.0 UIs and then the advantages of Web 2.0 UIs.

2.1 Web 1.0 UI for the hotel booking example

A typical solution using a Web 1.0 UI is shown in Figure 2. It has been extracted from a real running application[1] and follows the usual booking steps which go as follows:

- Step 1: selection of check-in and check-out dates (Figure 2.1)

[1] http://www.booking.com

- Step 2: room type selection (Figure 2.2)
- Step 3: payment and personal information (Figure 2.3)
- Step 4: summary and confirmation form (not depicted in the Figure)

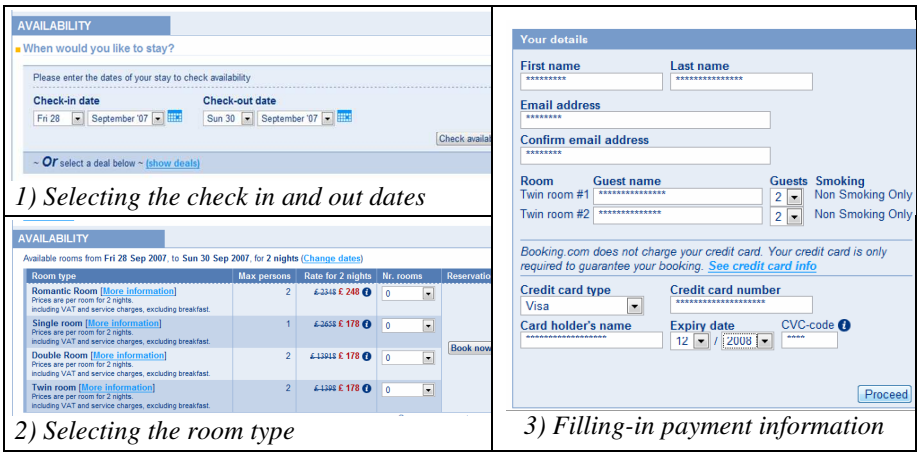

Figure 2: Different steps in the hotel booking example Web 1.0 application

This is shown as general example of multi-step tasks which normally require an HTML page for each step. The main problems regarding the four main aspects are the following:

Data. The way in which the contents are distributed between clients and server is quite limited. While the storage of persistent and volatile content on the server is possible, the storage of content on the client is quite limited [Bozzon, 06]. In the booking example, it would be better to have the possibility of storing volatile data at the client side (e.g., date selections and number of rooms). This would minimize the bandwidth usage and improve user experience. However, this is not possible with Web 1.0 UIs.

Business Logic. Processes are executed only on the server: the client performs a request and the server builds a new page that is sent to the client as a response. In the booking example, this could be improved by performing some complex operations at the client side (e.g., available date validation). Once again this would minimize the bandwidth usage again and decrease server process loading. However, this is not possible with Web 1.0 UIs.

Presentation. In Web 1.0 applications the presentation capabilities are quite limited. Web 1.0 application renderers are not able to provide multimedia native support and they need plug-ins in order to be able to play video and audio at the client side (e.g., www.youtube.com). Another limitation is that they do not support rich interactions and UI animations. Furthermore, the presentation to the user could be undesirably heterogeneous due to the different rendering engines of the browsers. To obtain these features expensive JavaScript and CSS browser-dependent code needs to be written without a guarantee that the code will deal with future browsers updates. The booking example can also be improved by performing richer interactions. However, this is not possible with Web 1.0 UIs.

Communications. Web 1.0 applications allow only synchronous connections and the communications are always originated at the client side. In the booking example, it could be improved by having asynchronous communications allowing partial refreshments of the UI and once again minimizing the bandwidth usage. However, this is not possible with Web 1.0 UIs.

2.2 Web 2.0 UI for the booking example using RIA

Let us consider now the same example but with an application developed using RIA technologies. The example has been taken from *www.ihotelier.com*. Figure 3 shows a snapshot of the Web 2.0 UI from the booking process[2]. The main difference is that now a single-page paradigm [Mesbah, 06] is followed. So, all the steps of the task can be performed in a single Web page.

Figure 3: RIA with a single page for the booking task

The main advantages of Web 2.0 approach are the following:

Data. Through RIAs, client's memory is available to be used by the application; the amount of client storage capacity depends on the selected RIA technology and the user's preferences. In a RIA it is common to store persistent and volatile content on the client (e.g., a shopping cart or a calendar of appointments which can be stored locally on the client), to manipulate it (e.g. filtering, adding, deleting or modifying ordered items or appointments) and to send the manipulated content to the server once the whole operation has been completed.

Business Logic. RIAs have a different navigation structure from Web 1.0 applications. They operate as single page applications, where the nested pages are able to be processed either by the client or by the server. Due to the augmented process

[2] https://reservations.ihotelier.com/onescreen.cfm?hotelid=6193&languageid=1

capability of the client, in RIA both the client and the server can carry out complex operations (e.g., data filtering or numeric operations for the booking example).

Presentation. RIAs offer new functionalities that improve presentation and user interactions (e.g. native multimedia support for showing rooms in other booking applications, etc). RIAs avoid unnecessary full UI refreshments and allow progressive presentation loading when needed. This would improve the bandwidth usage while maintaining homogeneous presentation in the booking example.

Communication. RIAs allow both synchronous and asynchronous communications. Distribution of data and functionality across client and server broadens the features of the produced events as they can originate, be detected, notified, and processed in a variety of ways. In this sense, pulling and pushing communication capabilities are available and developers may use them depending on the Web application nature. For the booking example it would then be possible to obtain partial refreshments of the UI.

Summarizing, by using the single page application paradigm RIA is able to overcome the limitations of the Web 1.0 that we have shown in this section. Once the main benefits of using RIA for developing Web 2.0 have been introduced, next we present a method for the systematic adaptation of existing Web 1.0 UIs to Web 2.0 UIs.

3 RUX-Method Overview

RUX-Method overview is depicted in Figure 4. Here we only give an overview with the aim of understanding the adaptation phases. The reader is referred to [Linaje, 07] for details. Due to it being a multidisciplinary proposal and in order to decrease cross-cutting concepts, the UI specification is divided into levels. According to [Limbourg, 07] a UI can be broken down into four levels, Concepts and Tasks, Abstract Interface, Concrete Interface and Final Interface.

Following this approach RUX-Method method takes Concepts and Tasks (i.e. data and business logic) from the underlying legacy Web model while the rest of the Interface levels are mainly based on RUX-Method Interface components. Thus, RUX-Method Interface levels are mainly made up of Abstract Interface Components, Concrete Interface Components and Final Interface Components.

In RUX-Method the Abstract Interface provides a UI representation common to all RIA devices and development platforms, without any kind of spatial, look&feel or behaviour dependencies, so any device that runs RIAs has the same Abstract Interface. The components of the Abstract Interface are independent from specific devices and rendering technologies. Here, we talk about different abstract components: containers (e.g. views), contains (e.g. text, video, images, etc.) and connectors (the connection with the business logic of the underlying Web model).

Then, in the Concrete Interface we are able to specify the Abstract Interface for a specific device or group of devices. This Interface level allows achieving better user experiences at run-time optimizing the UI through an ad-hoc design approach. Concrete Interface is divided into three Presentation levels: Spatial, Temporal and Interaction Presentation. The Spatial Presentation allows the specification of the spatial arrangement of the UI as well as the look&feel of the Interface components. The Temporal Presentation allows the specification of behaviours which require a

temporal synchronization (e.g. animations). The Interaction Presentation allows the specification of the user's behaviour with the RIA UI.

RUX-Method ends with the Final Interface which is specific for a device or a group of devices and for a RIA development platform. From a practical point of view, RUX-Tool provides the code generation of the modelled application. This code is then ready to be deployed.

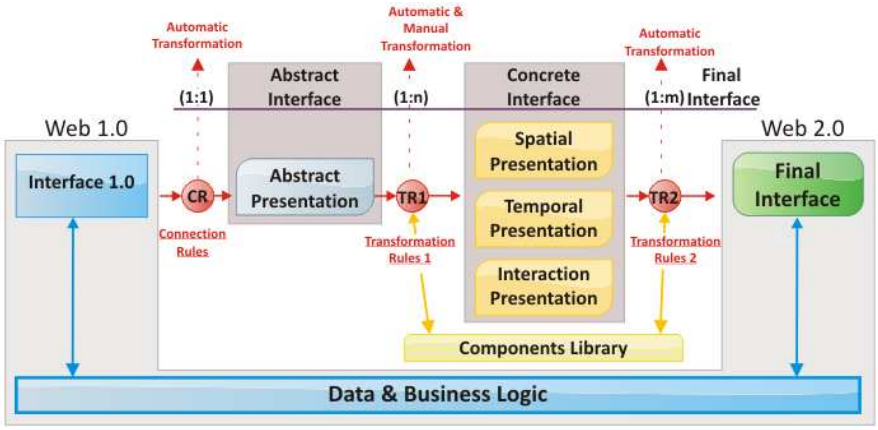

Figure 4: RUX-Method architecture overview

4 Adaptation phases in RUX-Method

There are two kinds of adaptation phases in the RUX-Method according to the Interface levels defined above. Firstly, the adaptation phase that catches and adapts Web 1.0 data, contents and navigation to RUX-Method Abstract Interface is called *Connection Rules*. Secondly, the adaptation phase that adapts this Abstract Interface to one or more particular devices and grants access to the business logic is called *Transformation Rules 1*.

Closely related to the Transformation Rules 1, is the Component Library (see Figure 4). Each RUX-Method Interface level is composed by Interface Components whose specifications are stored in the Library. The Library also stores how the transformations among components of different levels are carried out.

Finally, there is an additional transformation phase that is not an adaptation one. This phase completes the MDA life-cycle of RUX-Method supporting and ensuring the right code generation (*Transformation Rules 2*). In order to provide the most complete vision for the reader of the adaptation method and the adaptation phases, Transformation Rules 2 will be treated also in this section.

4.1 First adaptation phase: Connection Rules

The first adaptation phase, marked CR in Figure 4, extracts all the relevant information (data model and business logic) from the adapted Web model to automatically build a first version of the Abstract Interface. This process is performed automatically because Connection Rules establishes the way the matching takes place

among the previous Web model elements and the Abstract Interface components. Notwithstanding, the designer can refine this automatic version of Abstract Interface to suit their needs (e.g. content grouping).

Usually, Web models have graphic (based on E-R or UML) and textual (using XML based syntax) notations to represent their concepts intuitively. The connection mechanism graphically expresses the set of relationships between each element in the Web model (e.g. container, unit) and the potential target component in the Abstract Interface of RUX-Method (e.g. view, media).

The connection process starts selecting the set of Connection Rules to be used. This selection depends on the Web Model used for the development of the application to be adapted. A set of Connection Rules exists for each potential Web Model being considered (e.g. WebML, OOHDM, UWE, OO-H, among others).

The way in which the Connection Rules algorithm works depends on the Web model used and it is reusable in any other project using the same Web model. The following information is extracted from the previous Web model:

- The connections between pages, which allow us to know content grouping and to remain this grouping information context.
- The data used by the Web application and its relationships.
- The existing hypertext element groupings.

This information will also grant the Concrete Interface access to the "*operation chains*" or "*operation points*", which represent the operational links in the hypertext navigation models.

After the Connection Rules are applied, we obtain an Abstract Interface with components which are independent from the platform and the final rendering device.

4.2 Component Library

In order to understand the way in which the Transformation Rules are applied in the second and third steps (marked *TR1* and *TR2* in Figure 4), we need to know the role played by the Component Library.

The Components Library is responsible for: 1) storing the component specification (name, methods, properties and events), 2) specifying the transformation/mapping capabilities for each component from an Interface level into other components in the following Interface level and 3) keeping the hierarchy among components at each Interface level independently from other levels. The set of Interface components defined in this library can be increased or modified by the designer according to the specifications of the project. In addition, the set of available transformations/adaptations for each Component can be 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 platform.

The Components Library is partially based on XICL (eXtensible user Interface components Language) [Sousa, 04]. XICL is an extensible XML-based mark-up language for the development of UI components in Web applications. However, RUX-Method extends XICL to define Interface components (properties, methods and so on) and to establish mapping options among Interface components of adjacent Interface levels. Figure 5 illustrates the Entity-Relationship diagram for the Component Library.

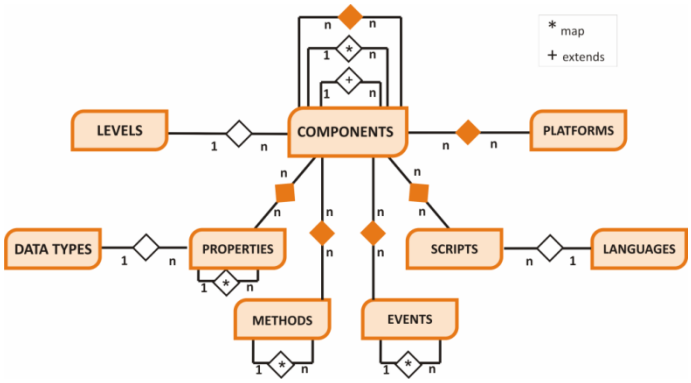

Figure 5: Component Library E-R description

4.3 Second Adaptation phase: TR1

In the second adaptation phase, marked as TR1 in Figure 4 a draft version of the Concrete Interface is obtained from the Abstract Interface. This is done automatically due to TR1 establishing the matching between Abstract Interface components and Concrete Interface components. This version can be refined by the designer to suit their needs (e.g. changing the spatial arrangement, adding temporal behaviours).

TR1 establishes the correspondences which are allowed among Abstract Interface Components and Concrete Interface Components. The set of different components which can form the Abstract Interface is fixed and limited by a number of elements set by RUX-Method as native. However, the set can be conceptually increased by extending the Component Library. The size of Concrete Interface Components set is variable and depends on the number of RIA elements to be defined dynamically in the Component Library. Following these criteria, the set of Transformation Rules 1 is also dynamic, so if the designer includes or modifies a Concrete Interface Component, the necessary changes in TR1 must be done. Any other way, RUX-Method skips that Component which will not be available at modelling time.

4.4 Transformation phase: TR2

Finally, in the last phase, TR2 (Transformation Rules 2), the Final Interface is automatically obtained depending on the chosen RIA rendering technology [Brent, 07] (e.g. Laszlo, Flex, AJAX, XAML). This process is performed automatically because TR2 establishes the way the matching takes place among Concrete Interface components and Final Interface components.

This case is different from the previous one. In TR2, both the set of origin Components (Concrete Interface) and the set of target Components (Final Interface) can be dynamically changed in the Component Library. On the one hand, a Concrete Interface Component is defined only once in the Library. On the other hand, there can be many specifications of each Final Interface Component in the Library according to the quantity of available rendering platforms. As an example, there is only one “window” Concrete Interface specification, while there is one “window” Final Interface specification for AJAX, another one for FLEX and so on (see Figure 5). In

this sense, the Concrete Interface is a DSL (Domain Specific Language) for the RIA domain, independently of the technology specification.

As in the previous adaptation phase, the mapping of methods, events and properties is done individually. The set of Transformation Rules depends on the specification of the Concrete and the Final Interface Components, so updating a Component in one of these Interface levels requires updating the Transformation Rules 2 according to the target platforms where we want to deploy the Final Interface.

5 Motivating example revisited

In this section we show how to use RUX-Method to adapt a Web 1.0 UI to a new Web 2.0 multidevice UI. For this purpose the motivating example shown in Section 2 will be used.

First of all, the Web 1.0 application depicted in Figure 2 must be modelled using a Web Model. The selected Web Model is WebML [Ceri, 02] which has been used in the past with RUX-Method [Preciado, 07]. The modelled application is our start point. Then, the second step is the adaptation of Web 1.0 UI.

It is not the objective of this paper to give a full overview of WebML. We will only give the necessary details to understand the adaptation method.

5.1 The hotel booking example with WebML

There are several proposals for conceptual modelling of Web applications. These proposals allow reasoning at a high level of abstraction without committing to detailed architectural and implementation issues. WebML is one of these proposals and its distinguishing feature is an extensive use of diagrams which provide a very intuitive design to the designer.

According to the WebML approach, Web applications are mainly composed by two orthogonal concepts: the data model which describes the schema of data resources according to the ER model and the hypertext model which describes how data resources are assembled into information units and pages, and how such units and pages interconnect to constitute a hypertext.

WebML provides modelling abstractions and its CASE tool called WebRatio can transform these abstractions into specific pages. Figure 6 depicts part of the data model and part of the hypertext model of the booking example in order to detail the way the application goes. The E-R diagram needs no description since it is well-known and frequently used in many Software Engineering disciplines. The business process goes as follows:

The user enters into the Web application through its homepage (*Date selection* page) where he or she fills in the dates form. When the form is sent to the server, the Web application selects those rooms which follow the criterion of not being booked on the selected dates (*Available room's* selector unit). This data selection is used by the user to choose the room type. Only those types where there is at least one free room are shown (*Room selection* index unit). When the user selects a room type the server then calculates the total amount to charge (*MathUnit1*). This is passed as result to a form where the user enters his personal and payment information (*payment form*). When the user sends this information to the server, a summary page including a form

to confirm all the booking details (*Conformation form*) is sent from the server. Finally, the user must confirm the booking and a transaction is triggered to create a guest and a booking (*Data insert*). Using the WebML CASE tool, designers can generate the application shown in Figure 2 after some styling specifications.

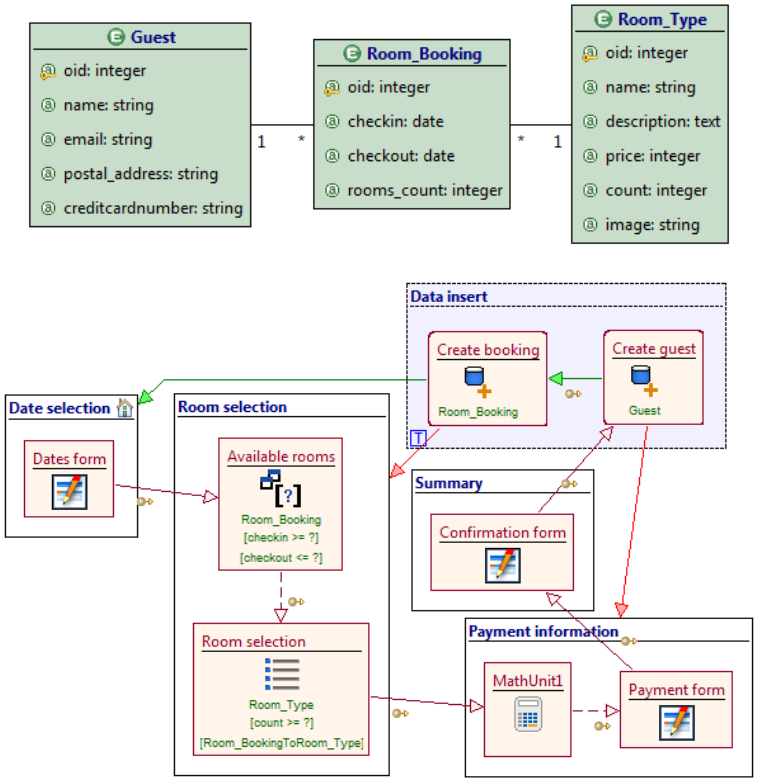

Figure 6: Motivating example using WebML: top) data; bottom) hypertext model

5.2 Adapting the hotel booking example toward Web 2.0 UI

The first phase of RUX-Method is applying the Connection Rules. Figure 7 depicts how this adaption is carried out for the WebML specific case.

The Connection Rules for WebML filter the information offered by WebRatio, obtaining only the information needed to be used for building Abstract, Concrete and Final Interface. This information refers mainly to the <Structure> and <Navigation> elements, avoiding the remaining information regarding Presentation, Mapping and Localization among others.

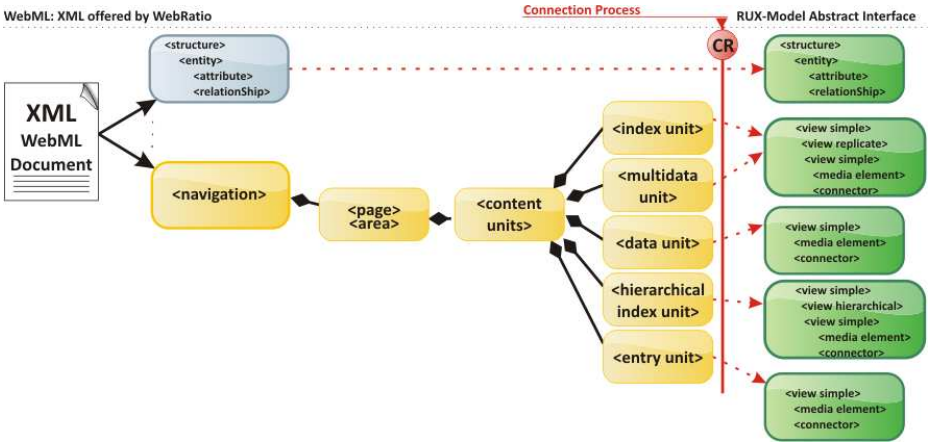

Figure 7: Connection Rules for the WebML specific case

The WebML to RUX-Method connection process is performed surfing top-down the WebML hierarchy, matching each element according to the relation pattern established by the RUX-Method Connection Rules. The main idea is to recognize the WebML concepts and to classify the potential correspondences with the RUX-Method Abstract Interface elements.

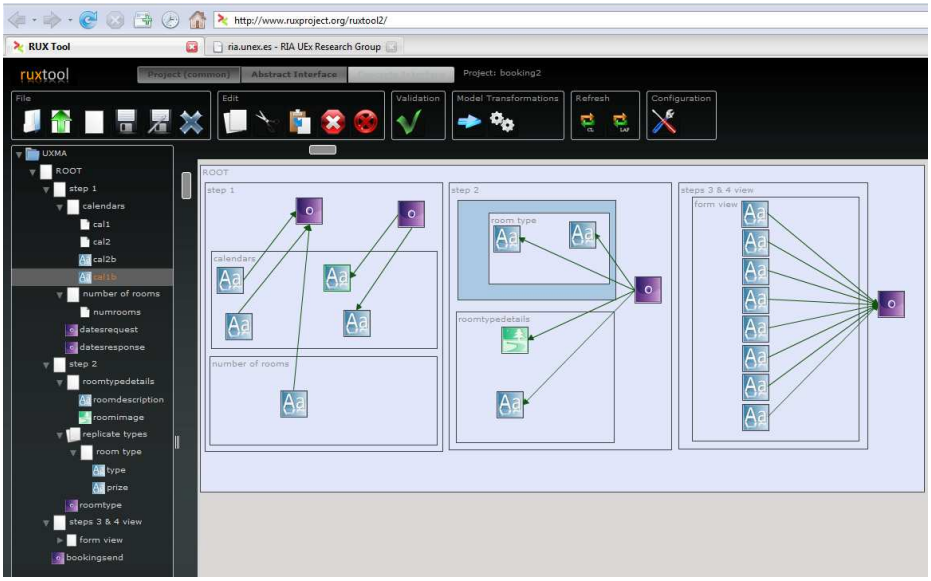

Figure 8: RUX-Method Abstract Interface for the booking example.

Figure 8 represents a RUX-Tool screenshot and shows the RUX-Method Abstract Interface automatically obtained once the Connection Rules have been applied over the legacy Web 1.0 application design. The black colour in the background belongs to the tool. The abstract interface consists of the different boxes and arrows.

The designer can refine this Abstract Interface using RUX-Tool (on-line application also shown in Figure 8) when necessary. Once the Abstract Interface has been obtained, the designer can apply the Transformation Rules 1 constraining the Interface design process through the selection of many features available in the RIA capable devices (type of multimedia files support, screen size restrictions...)

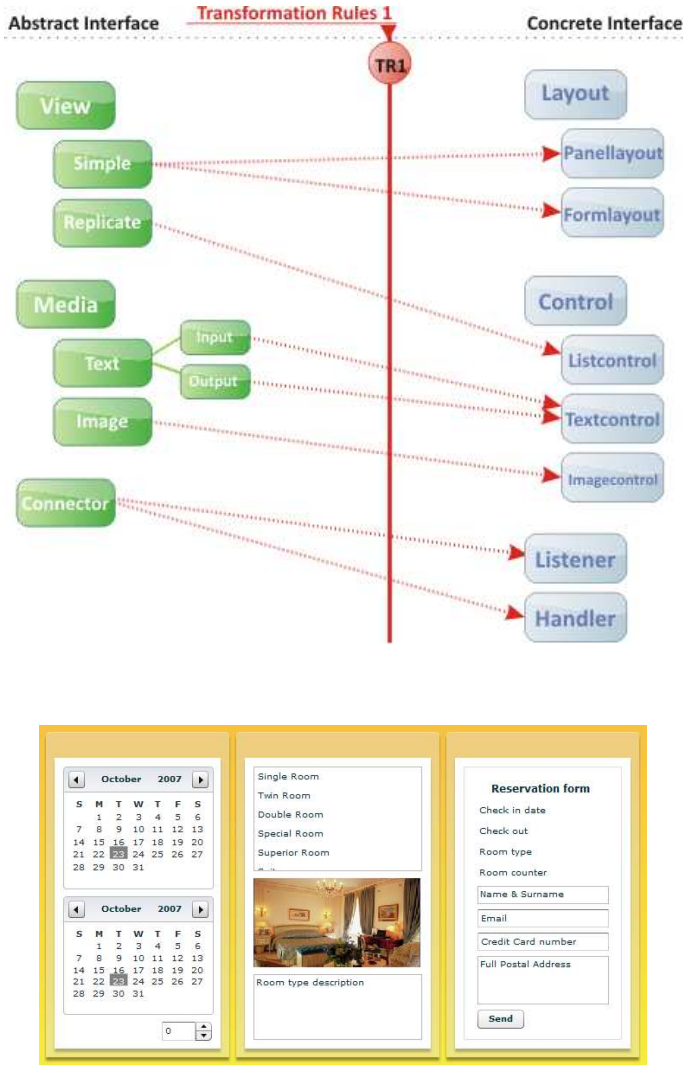

Figure 9: top) RUX-Method Transformation Rules 1 for the motivating example. bottom) RUX-Method Concrete Interface for the motivating example.

After the target device(s) selection for the application, the designer automatically obtains the Concrete Interface. The type of the Concrete Interface components obtained and the availability of the rest of the components defined in the Component Library is constrained by the chosen device(s) (i.e. it is not the same menu for a PC than for a smartphone). This Interface must be refined to obtain an application similar to Figure 3, but without any kind of rendering technology dependency. Figure 9 (top) depicts the Transformation Rules 1 schema for the motivating example. In this schema we focus on the origin (Abstract Interface components), the destination (Concrete Interface components) and the mapping relationships among them. Through the application of these rules, the Concrete Interface is obtained and after a manual refinement designers can achieve their goal as depicted in Figure 9 (bottom).

The final phase of the RUX-Method adaptation method is the application of Transformation Rules 2. Applying this set of rules, the designer obtains the Final Interface level which ends with the code generation when using the RUX-Tool. In this way an application is obtained like that shown in Figure 3. These Transformation Rules allow the designer to choose the final rendering platform. Figure 10 extends Figure 1. The reader can see how the old UI of the Web application can be adapted to a new Web 2.0 multidevice UI. In Figure 10, the Abstract Interface is transformed into two Concrete Interfaces, one with a special arrangement customized for PCs and the other for a group of mobile devices. One Concrete Interface is transformed into two Final Interfaces for a PC using different rendering technologies and the other Concrete Interface into two Final Interfaces for different devices with the same rendering technology. This is the way in which multidevice UI design capabilities are obtained while UIs remain highly customizable for each device taking full advantage of the devices' capabilities (e.g. screen size).

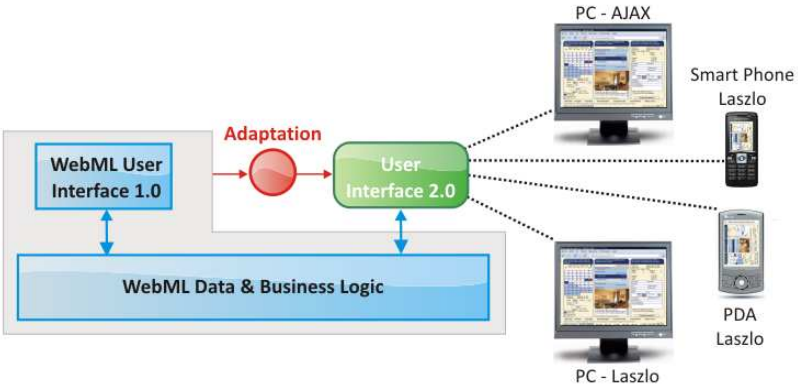

Figure 10: Multidevice adaptation

6 Conclusions and future work

This paper introduces RUX-Method, a Model Driven Method for the systematic design of RIA UIs adapting existing HTML-based Web Applications to provide them with multimedia support, offering more effective, interactive and intuitive user experiences.

To our knowledge this is the only Web Engineering approach able to adapt existing Web 1.0 applications based on models to new Web 2.0 "rich" UIs. This adaption is performed with no changes in the business logic already modelled.

Conceptually, RUX-Method can be used on several Web development models. The use of Web models is mandatory because RUX needs a full understanding of the legacy Web application, and the best way to do this is by means of models. At the implementation level, RUX-Tool has a series of prerequisites regarding the models that can be used in order to automatically extract from them all the information stored by these models (mainly a declarative representation of the model).

Currently, RUX-Tool works together with WebML (using WebRatio, the WebML CASE tool), but there are works in progress with UWE [Koch, 02] and OO-HDM [Urbieta, 07].

Acknowledgements

This work was partially funded by PDT06A042 and TIN2005-09405-C02-02.

References

[Bozzon, 06] Bozzon, A., Comai, S., Fraternali, P., Toffetti, G.: Conceptual Modeling and Code Generation for Rich Internet Applications, International Conference on Web Engineering (ICWE), Springer, 353-360, 2006

[Brent, 07] Brent, S.: XULRunner: A New Approach for Developing Rich Internet Applications, Journal on Internet Computing, IEEE, vol.11, iss. 3, 67 - 73, 2007

[Canal, 06] Canal, C., Murillo, J.M., Poizat, P.: Software Adaptation, Coordination and Adaptation Techniques, L'Object, vol. 12, iss. 1, 2006

[Ceri, 02] Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications, Morgan Kauffmann, 2002

[Daniel, 07] Daniel, F., Yu, J., Benatallah B., Casati, F., Matera M., Saint-Paul, R.: Understanding UI Integration: A Survey of Problems, Technologies and Opportunities, Journal on Internet Computing, IEEE, vol. 11 iss. 3, 59-66, 2007

[Koch, 02] Koch, N., Kraus, A.: The Expressive Power of UML-based Web Engineering, Int. Wsh. Web-Oriented Software Technology (IWWOST), 105-119, 2002

[Limbourg, 07] Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Lopez, V.: UsiXML: a Language Supporting Multi-Path Development of User Interfaces, 9th IFIP Working Conference on Engineering for HCI, LNCS Springer-Verlag, vol. 3425, 207-228, 2005

[Linaje, 07] Linaje, M., Preciado, J.C., Sánchez-Figueroa, F.: Engineering Rich Internet Application User Interfaces over Legacy Web Models. Internet Computing Magazine, IEEE, vol.11, no.6, pp.53-59 (2007)

[Mesbah, 06] Mesbah, A., van Deursen, A.: Migrating Multi-page Web Applications to Single page Ajax Interfaces, Delft University of Technology SERG, TUD-SERG-2006-018, 2006

[Preciado, 05] Preciado, J.C., Linaje, M., Sánchez, F., Comai, S.: Necessity of methodologies to model Rich Internet Applications, IEEE, Internet Symposium on Web Site Evolution (WSE), pp. 7-13, 2005

[Preciado, 07] Preciado, J.C., Linaje, M., Comai, S., Sánchez, F.: Designing Rich Internet Applications with Web Engineering Methodologies, IEEE, International Symposium on Web Site Evolution (WSE), pp. 23-30, 2007

[Sousa, 04] de Sousa, G., Leite, J.C.: XICL - an extensible markup language for developing user interface and components, International Conference on Computer-Aided Design of User Interface, vol. 1, 2004

[Suh, 05] Suh, W.: Web Engineering: Principles and Techniques, IGI Publishing, Christodoulou at al.'s chapter, 31-75, 2005

[Urbieta 07] Urbieta, M., Rossi, G., Ginzburg, J., Schwabe, D.: Designing the Interface of Rich Internet Applications, In Almeida, V., Baeza-Yates, R. (eds.) Proceedings of the 5th Latin American Web Congress, pp.144-153, IEEE (2007)

[Yahiaoui, 04] Yahiaoui, N., Traverson, B., Levy, N.: Classification and Comparison of Adaptable Platforms, I WCAT Workshop, Available at http://wcat04.unex.es/, 55-61, 2004