Designing Rich Internet Applications with Web Engineering Methodologies

Preciado, J.C.; Linaje, M.; Comai, S.; Sánchez-Figueroa, F.
1Quercus Software Engineering group. Universidad de Extremadura (10071, Cáceres, Spain)
2Politecnico di Milano. Dipart. di Elettronica e Informazione (20133, Milano, Italy)
{jcpreciado; mlinaje; fernando}@ruxproject.org; sara.comai@polimi.it

Abstract

Nowadays, Rich Internet Applications are gaining ground thanks to the facilities they provide to develop Web applications with multimedia, high levels of interactivity, collaborative work, and/or homogeneous presentation requirements at the client side. However, this new kind of Web applications currently lacks complete methodologies and models which aid its design and development. This paper introduces the concepts at the base of Rich Internet Applications to take full advantage of their new capacities, and proposes an integrated Web Engineering approach based on the WebML and the RUX-Model conceptual models for supporting a high-level design of these applications and their automatic code generation.

1. Introduction and motivation

Currently, the complexity of tasks performed through Web applications is increasing, in particular when high levels of interaction, client-side processing, and multimedia capacities have to be provided. In this context traditional HTTP-HTML Web applications are showing their limits and developers are building the future of the Web using Rich Internet Applications (RIAs) technologies, which are Web applications with many additional features.

RIAs offer online and offline capabilities, sophisticated user interfaces, the possibility to store and process data directly on the client side; they offer high levels of user interaction, usability and personalisation, minimise bandwidth usage, and separate presentation and content at the client side.

The work in [6] showed the limitations of methodologies coming from the Web, Multimedia and Hypermedia fields to model these new capabilities offered by RIAs. Among the different methodologies analysed in [6] the Web modeling language WebML [4] has been recently extended [3] [9] to achieve more RIAs design objectives. In addition, the RUX-model has been proposed in [7] to cope with further uncovered RIAs capabilities.

The objective of this paper is twofold: to identify the main concepts that need to be taken into account in the design of Rich Internet Applications and to show how these concepts can be captured by integrating two Web Engineering proposals WebML [3] and the RUX-Model [7]. However, the topics treated here have a general validity not limited to the use of WebML and RUX-Model. Other design related methodologies could similarly be extended to support RIA capabilities.

The rest of the paper is structured as follows: in Section 2 we identify the concepts to be considered when designing RIAs; in Section 3 we show how these concepts have been included in WebML and in the RUX Model. Section 4 estimates the suitability of the two models with respect to the proposed approach. Finally, in Section 5 conclusions and future work are presented.

2. Identifying concepts for designing RIAs

RIA technologies provide new/additional capacities with respect to traditional Web applications that must be taken into account in their design.

Designing RIAs with Web engineering methodologies requires adapting the Web development flow of traditional Web applications to consider the new client-side capacities, the new presentation features, and the different communication flows between the client and the server. In this paper, four phases characterizing RIAs are analyzed (see Figure 1): data modeling, business logic modeling, presentation modeling, and communication modeling. Such phases require extending the models of traditional Web applications.

In Web methodologies the phase in charge of designing the tasks is commonly known as hyper-text/navigation. However, RIAs do not use hyper-text/navigation in the way traditional Web applications do. So, in order to provide a wider perspective we call this phase business logic design. Moreover, a new phase becomes relevant in RIAs: the communication design. This concern crosses data, business logic and presentation.
In the following subsections we analyse in detail these design phases: an initial overview of each phase is provided, the possible problems that may arise are highlighted, the main concepts to be taken into account are described, and design guidelines are given.

2.1. Data foundations

Overview. In traditional Web applications the way in which the contents are distributed between clients and server is quite limited. Focusing on data-intensive Web applications, the storage of persistent and volatile content on the server is possible, while the storage of content on the client is quite limited [3].

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 or on the client preferences. In a RIA it is common to store persistent and volatile content also on the client (e.g., the shopping cart in an e-commerce application or a calendar of appointments could be stored locally on the client), to manipulate it (e.g. add, delete or modify ordered items or appointments) and to send the manipulated content to the server once the whole operation has been completed. This situation also appears in other Software Engineering fields under the name of (temporal) disconnected applications [11].

Issues. The storage of data at the client side do not solely involves content distribution and content persistence, but also requires additional mechanisms to guarantee the data model integrity. Moreover, the designer must be careful about data security in the client, since sandboxes seem to be too insecure for sensitive data.

Description. The distributed data design specifies where the data reside and their duration, establishing what in [3] is termed levels of persistence. Four levels of persistence are identified embodying two dimensions: the distribution between the client and the server and the duration/persistence of the data. In a RIA the following levels should be considered:

- **Non-persistent (or volatile) at the client side:** data are available at runtime, but are lost when the user leaves the application (an example of data belonging to this level is represented by the runtime variables).
- **Persistent at the client side:** data remain stored on the client, even if the user leaves the execution of the application. In traditional Web applications cookies have been used for a long time but they provide very limited storage capabilities. RIAs provide new client-side storage facilities depending on the selected deployment platform (e.g., files, databases, or “shared objects”).
- **Non-persistent (or volatile) at the server side:** data reside on the server while the user is connected and disappear when the user leaves the execution of the application (they include, for example, server-side session variables).
- **Persistent at the server side:** data reside durably on the server, independently of whether the user leaves the application or not (e.g., in files and/or in databases).

When data are distributed, a typical problem is the maintenance of the information consistency between the client and the server. Taking into account this issue, we propose to specify different data granularity consistency levels. Three levels are proposed, covering the whole spectrum of RIAs necessities:

- **at field level:** when a particular field of a tuple is changed either on the client or on the server, this change is propagated to synchronize also the corresponding client and server data.
- **at tuple level:** when a tuple is changed either on the client or on the server, the changes are propagated to guarantee the client and server data consistency.
- **at packet level:** when a set of tuples (e.g., a products list) is changed either on the client or on the server and the content synchronization is triggered, these changes are propagated to synchronize also the corresponding client and server data.

Data changes and synchronization strategies can be specified using a platform independent information synchronization standard, like SyncML (Synchronization Mark-up Language) [8] where also the policy may be specified.

Guidelines. Initially, the designer can define a conceptual data model without taking into account the level of persistence of the data. For this task well-known approaches based on UML classes or Entity-
Relationship diagrams can be adopted. Then, taking into account also the next design phases, the designer can refine the data model by specifying the level of persistence of the data and the data granularity, according to the application necessities. In addition, when the designer chooses a storage method, he should also consider if special features of security are needed in order to avoid putting the sensitive data at risk (e.g., sensitive data can be kept only on the server).

2.2. Business logic foundations

Overview. In traditional Web applications 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 response.

RIAs have a different navigation structure from traditional Web applications. They operate as single page applications [2], where the nested pages are able to be processed by the client or by the server. Due to the augmented process capability of the client, in RIA both the client and the server can carry out complex operations (e.g., data filtering, numeric operations, etc.).

Issues. The complexity of the tasks that can be performed in a RIA require a richer kind of distributed functionality with respect to traditional Web applications. Moreover, additional mechanisms are needed to start a conversation (request) from both sides (client and server). This last aspect will be considered in Section 2.4.

Description. RIAs allow distribution of the business logic. This situation does not solely include the distribution between client and server tasks, but also a new kind of task called mixed. The business logic can be therefore performed:

- **Client side**: when all the logic required for a particular task is performed only on the client.
- **Server side**: when all the logic required for a particular task is performed only on the server.
- **Mixed**: when the logic is distributed between the client and the server, in such a way that to complete a complex task, one part is carried out on the client and another part on the server.

Client side business logic is more adequate to take full advantage of the client capabilities available when using RIA technologies, while server side business logic is typically used in traditional Web applications when client capabilities are quite limited.

Mixed business logic is necessary when the business logic at the client side needs to connect to the server to process some operations (e.g., when the server side application state has changed and the connected clients must be notified or when operations require high security levels).

Guidelines. Conceptually, a task is independent of its distribution. The designer can face this design phase in two steps: first, he can specify the business logic functionality without considering the distribution between the client and the server; then, he can refine the specification, by choosing where and how these tasks have to be carried out.

Notice that in this way the design of the business logic can be done at a higher level of abstraction and that design reusability can be exploited, in particular, when UML diagrams or specific design specification languages for modelling Web applications are used.

Since a strong relationship between the data and the business logic exists (the application will publish the data stored in the data layer), the distribution of a task is also constrained by the distribution of the data used by the task: if server data need to be accessed or manipulated, at least part of the business logic must be managed by the server.

2.3. Presentation foundations

Overview. Presentation in traditional Web applications is quite limited. On the one hand, traditional Web application renderers are not able to provide multimedia native support and they need plug-ins in order to be able to show video and audio homogeneously at the client side (e.g., to show the content of the www.youtube.com site). On the other hand, they do not support rich interactions (e.g., drag&drop) and animations: to obtain them expensive JavaScript browser-dependent code needs to be written. RIAs offer new functionalities that improve presentation and users interactions.

Issues. Presentation in RIAs focuses on the layout, on the definition of styles, and on capturing the behaviors of the application and the interaction of the users. However, for managing the different aspects, RIA UIs are very dependent on the device where they are going to be rendered. The device establishes constraints like screen size and multimedia support and this should be taken into account to adapt the UI for achieving better user experiences.

Description. Due to the complexity of representing the presentation concepts involved in the RIA development a separation of concerns is required. The following topics have been identified:

- **Spatial arrangement** of the UI: it specifies the positioning, the dimension, and the look&feel of the Interface elements in the application space.
- **Temporal behaviours**: they define the temporal logic relations among the interface elements, without considering the intervention of the user. The validation of the set of temporal relations al-
lowed by the application can be based on multimedia standards (e.g., SMIL [12]). This aspect is very useful in order to represent behaviours that take place temporally on the Interface elements or to specify calls to the underlying business logic using time events.

- **Interaction behaviours**: they capture the interaction of the user (e.g., a mouse click on a presentation element) and specify the reaction to such interaction.

**Guidelines.** Conceptually, the designer can treat the three aspects of presentation design in two steps: first, he can specify the set of presentation elements and their spatial arrangement, colour, dimension; then, he can design the reactions of each presentation element to each temporal or interactive behaviour. The presentation elements could be associated with one or more reactions to be launched at the same time or at different times. The final presentation design can have complex set of behaviours over a presentation element: in such a case the designer must define the reaction rules carefully, in order to ensure the presentation consistency and to avoid contradictory behaviors.

Several sets of behaviours over the same group of presentation elements could be defined. This allows to better support the deployment of the same application over different types of devices, each one with different presentation capacity. For example, a shopping cart application could have the same group of presentation elements with different behavioural responses if it accessed using a PC (where, for example, both a simple bottom and the drag-and-drop could be used to add items to the shopping cart) or a PDA (where only a simple bottom can be provided to add items to the shopping cart).

### 2.4. Communication foundations

**Overview.** Traditional Web applications allow synchronous connections and the communications are originated at the client side.

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.

**Issues.** Communication is a cross-cutting concern (Figure 1) related to data synchronization, business logic distribution and presentation.

**Description.**

- Communication and Data foundations: it is mainly related with data consistency which can be invoked by the client (pulling mode) or by the server (pushing mode) for any kind of event.

Synchronization granularity is closely related to data granularity described in Section 2.1. Regarding data access three different alternatives, coming from Software Engineering, come into sight [11]: message-based (based on a message queue that is generated at one side and it is processed at the other side), state replication (based on a modified rows log) and method replay (based on operations recorded on the one side which are replayed in the other side). It is very usual to provide serialization and deserialization mechanisms on the server and the client. Such mechanisms increase data processing time but decrease bandwidth usage for data-intensive RIAs.

- Communication and Business Logic foundations: it deals with the communication among distributed tasks. It is closely related to mixed business logic described in Section 2.2. Client applications have to be notified of events occurring outside their execution environment, either triggered by other clients or by the server. This is useful to carry out business logic applications like collaborative RIAs, using pulling/pushing, synchronous/asynchronous methods.

- Communication and Presentation foundations: presentation uses the single page application paradigm. In order to decrease the initial loading and bandwidth usage, it is necessary to distribute the application presentation loading over the application runtime.

The communication must provide the binding between the presentation and the underlying data/business logic layers using synchronous/asynchronous methods.

**Guidelines.** Regarding the synchronization for data consistency the designer should choose the best granularity level and serialization usage according to the amount of information being communicated at runtime.

Regarding collaborative RIAs, clients store the state of the application mainly using “volatile” data. In collaborative RIAs, business logic triggers this volatile data synchronization among many clients using pulling and pushing capabilities. The designer may use different communication mechanisms for this issue, taking into account the next guidelines: 1) The method relay is less common than the message-based one, which is a wide-spread approach in many RIA technologies; 2) The message-based alternative has many advantages as good fault-tolerance, quick response to asynchronous events, and minimization of the bandwidth usage [11] that affects RIA performance in the clients too.

In mixed business logic, tasks should be grouped to minimize the client/server roundtrips and to allow...
transactional blocks in order to guarantee a simpler recovery in case of faults [3].

The designer must be able to establish loading priority levels in order to provide progressive presentation downloading according to the required user experience.

3. Designing RIAs with WebML and RUX

In this section we show how part of the concepts described in Section 2 have been captured by the WebML modeling language extended to cope with data and business logic distribution (Bozzon, Comai, Toffetti & Fraternali) [3] and by the RUX-Model (Rich User eXperience Model), focussing on the presentation design of RIAs (Linaje, Preciado & Sánchez-Figueroa) [7]. Both models are supported by a CASE tool: Webbratio and RUX-Tool, respectively.

3.1. WebML data RIA modeling

WebML allows specifying at the conceptual level a Web application publishing or manipulating data. Content is modeled using Entity-Relationship (E-R) or UML class diagrams. To support RIA data modeling, the WebML data model has been extended with the specification of the levels of persistence: the designer can explicitly specify if an entity or a relationship is stored on the server or on the client (an “S” or a “C” symbol are used to mark server-side and client-side entities/relationships, respectively), permanently or temporarily (a filled or an empty icon are used, respectively). Constraints apply in the specification of the possible combinations of persistence levels [3].

Figure 2 shows an example of RIA data schema, based on the Amazon Store demo application by Laszlo Systems [14]; it represents a Music Store repository, to offer functionalities for exploring and buying CDs. When not specified, by default entities are persistent at the server-side and relationships inherit the lower level of persistence of the entities they connect.

The following design decisions have been taken: a) the User, ShippingAddress, ShippingMethod, Order, and CreditCard entities are given permanent server-side persistence, because users must register permanently before issuing orders. Similarly, the CD, Review, and Track entities (and their IS-A descendants) are stored in the database; b) the shopping cart and the wish list must be created and/or manipulated at the client side; therefore, the entities Cart, CartItem, and Wishlist are given client temporary persistence; c) the client entities CartItem and Wishlist are connected by a temporary client relationship to the database entity CD, to enable looking up the full details of the CD stored in the database; d) the user may decide to save permanently his wish list and make it available to other users. Therefore, entity Wishlist is duplicated with permanent server-side persistence. In this way, the user will be able to create and modify the wish list at the client side, without invoking the server at each update. Conversely, the client entity Cart is not replicated on the server, because it lives only at the client. Instances of the client entity CartItem become instances of the server entity OrderItem, after check out.

3.2. WebML business logic RIA modeling

Upon the same data model, in WebML it is possible to define different hypertexts (called site views), targeted to different types of users or to different access devices. A site view is a graph of pages, possibly hierarchically organized into sub-pages. Pages comprise content units, representing components for content publishing: the content displayed in a unit typically comes from an entity of the data model, and can be determined by means of a selector, which is a logical condition filtering the entity instances to be published. Instances selected for display can be sorted according to ordering clauses. Units are connected to each other through links, which carry parameters and allow the user to navigate the hypertext. WebML also allows specifying operations implementing arbitrary business logic; in particular, a set of data update operations is predefined, whereby one can create/delete/modify the instances of an entity, and can be determined by means of a selector, which is a logical condition filtering the entity instances to be published. Instances selected for display can be sorted according to ordering clauses. Units are connected to each other through links, which carry parameters and allow the user to navigate the hypertext. WebML also allows specifying operations implementing arbitrary business logic; in particular, a set of data update operations is predefined, whereby one can create/delete/modify the instances of an entity, and create or delete the instances of a relationship.

To support RIA design the hypertext model of WebML has been extended with the explicit specification of distribution between the server and the client of its concepts: pages, content and operation units, selec-
tors, ordering clauses. Also in this model constraints apply in the specification of the possible combinations of server/client concepts [3]. Figure 3 shows an example extending the search function of the Amazon Store site. We model a Search page behaving as follows: the user can enter a keyword to be matched in the title of CDs for retrieving from the database an initial album list; then, the retrieved instances can be filtered by means of two conditions (price between a minimum and a maximum). Such filtering must occur at the client, without recomputing the list of albums fetched from the server.

In [9] WebML has been extended to explicitly model also event notifications between the client and the server or, through the server, between different clients, thus partially coping with the communication requirements. This work is inspired by generic distributed event systems and introduces new WebML primitives for generating, notifying and detecting events. Both pull and push modalities can be supported, in a synchronous or asynchronous way. For a complete description of the proposed approach the reader may refer to [9].

3.3. RUX-Model presentation RIA modeling

The RUX-Model UI specification is divided into three levels: Abstract Interface, Concrete Interface and Final Interface. In addition to the presentation concepts previously identified in Section 2.3, the RUX-Model provides reusable multi-device RIA UIs.

The Abstract Interface specifies the UI concepts common to all RIA devices and development platforms; no spatial arrangement, look&feel or behaviour are specified. The Abstract Interface elements include: i) Connectors, used to establish the relation between the UI component and the underlying data; ii) Media, representing atomic information elements that are independent of the client rendering technology: they have been categorized into discrete media (texts and images) and continuous media (videos, audios and animations); iii) Views, used to group the information that will be shown to the client at the same time. In order to group information, the RUX-Model allows the use of four different types of containers: simple, alternative, replicate and hierarchical views.

The Concrete Interface optimizes the UI for a specific device or group of devices. It is divided into three Presentation levels: Spatial, Temporal, and Interaction Presentation. The Spatial Presentation supports 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 those behaviours which require a temporal synchronization (e.g., animations). The Interaction Presentation models the user’s behaviour with the UI (e.g., pushing a bottom).

The Final Interface provides the code generation of the modeled application. This generated code is specific for a device or a group of devices and for a RIA development platform and it is ready to be deployed. The methods, events and properties mapping is done individually for each presentation element and target platform to be deployed in the Final Interface level: the transformation is performed using a specific set of
XSL transformation rules. For a complete description of the proposed approach the reader may refer to [7].

Fig 4. A snapshot of the RUX-Model Abstract Interface in the RUX-Tool

As an example, Figures 4 and 5 show the Abstract Interface and the Concrete Interface of the WebML CdList index unit depicted in Figure 3. In the RUX-tool a draft of the Abstract Interface of each element can be automatically derived by extracting all the relevant information from the data and business WebML models; the draft can then be refined by the designer. In the example of Figure 4 there is a simple view to contain the “Recommendations” section. This view contains another simple view including a connector (it is shown by means of a white circle) and a Replicate view with some media components according to the information that we need to show. The media component represents an atomic information element that is independent of the client rendering technology. Common media attributes are “source” and “connectorid”. These attributes are used to define dynamic data source of the media (obtained from the underlying Web model using a connector).

Fig 5. A snapshot of the RUX-Model Concrete interface in the RUX-Tool

From the Abstract Interface the RUX-tool can automatically derive a draft of Concrete Interface, that can then be refined by the designer. The example in Figure 5 shows the Spatial Presentation in RUX-Model, responsible for specifying the positioning, dimension and look&feel of the Interface Components in the application space. In this picture, the design of CD information is depicted: it is design only once the replication technique is applied depending on the dynamic list of CDs. The Spatial Presentation design is the first step in the Concrete Interface design. Over this Spatial Presentation RUX-Tool allows adding behaviours (temporal and interaction behaviours) over the presentation elements.

Once the Concrete Interface has been defined the Final Interface is automatically generated depending on the chosen RIA rendering technology (e.g. Lazeslo, Flex, Ajax, XAML).

4. Discussion

In this section we discuss the suitability of the WebML and RUX models to design RIAs, according to the RIA design requirements identified in Section 2. We define different coverage degrees where each degree shows how each phase (i.e., data, business logic, presentation and communication) is covered by each model. Four coverage degrees are available:

- Desired coverage degree, when the concept is ideally covered by the methodology.
- Partial coverage degree, when the concept is moderately covered.
- Limited coverage degree, when there is very little support towards the identified concepts.
- Zero coverage degree, when the concept is not covered at all.

Each Coverage Degree has a specific weight. In this evaluation the proposed weights are: Desired = 3, Partial = 2, Limited = 1 and Zero = 0.

Fig 6. Summarized discussion graphically

Figure 6 summarizes the results of the evaluation of WebML, its extension with RIA, the RUX-Model and their combination. Results are depicted textually and...
graphically: axis $x$ corresponds to the coverage degree, while axis $y$ represents each model.

The WebML extension for RIA is able to represent three of the four levels of persistence identified in the data foundations (data changes and synchronization strategies must be explicitly specified in the business logic model), obtaining a partial coverage degree. Then, it is able to represent the three business logic distribution types identified in the business logic foundations. Communication foundation is treated partially.

The RUX-Model includes all the topics identified in the presentation foundations, obtaining a desired coverage degree. Communication foundation is considered only in order to connect RUX-Model with the underlying data and business logic, obtaining a limited coverage degree. The topics related to data and business logic are not covered.

From the obtained results (compared with the ideal situation), we can deduce that at the moment it is not possible to fully cover the four phases required in RIA design using the extended WebML and RUX models. However, as can be seen from the results shown in Figure 6, the combination of the RUX-Model and the WebML RIA extension is approaching the goal of fully modeling RIA applications. Currently, this combination is the most complete proposal available in literature, although it does not fully cover all the identified concepts.

In addition to the possibility of specifying the models at a conceptual level, the combined use of Webratio (the WebML CASE Tool, available at http://www.webratio.org) and of the RUX-Model CASE Tool (RUX-Tool, available at http://www.ruxproject.org) supports also the automatic generation of RIA code [10].

### 5. Conclusions and future work

In this paper we have identified the foundation concepts implied in the design of Rich Internet Applications. We have also shown how these concepts are captured by two different Web engineering proposals (WebML and RUX-Model) and we have discussed the suitability of both models for covering the different RIA concepts.

To our knowledge there is no other model or combination of models able to cope with so many RIA capabilities. We plan to extend the current work in two different directions: on the one hand, we plan to combine the RUX-Model with other models (e.g. [1], [5] among others) different from WebML and, on the other hand, we plan to extend the WebML and/or the RUX-Model to fully cover the core concepts of the different RIA models identified in this paper.

### 6. References