Facing the Technological Challenges of Web 2.0: A RIA Model-Driven Engineering Approach

Francisco Valverde and Oscar Pastor

Centro de Investigación en Métodos de Producción de Software, Universidad Politécnica de Valencia, Spain
{fvalverde,opastor}@pros.upv.es

Abstract. One of the main reasons for the success of Web 2.0 is the improvement in user experience. This improvement is a consequence of the evolution from HTML User Interfaces (UI) to more usable and richer UI. The most popular Web 2.0 applications have selected the Rich Internet Application (RIA) paradigm to achieve this goal. However, the current Web Engineering methods do not provide the expressivity required to produce RIA interfaces. This work presents a RIA Metamodel to deal with the new technological challenges that have arisen with Web 2.0 development. This metamodel supports two main perspectives: 1) the definition of the UI as a combination of widgets from a selected RIA technology; and 2) the specification of the UI interaction as a consequence of the events produced by the user. In order to illustrate how this RIA Metamodel can be used in a Model-driven Engineering (MDE) method, this work also presents the integration of the RIA Metamodel with the OOWS method.

Keywords: Rich Internet Applications, Model-driven Development, Web User Interfaces.

1 Introduction

Nowadays, Web 2.0 is a topic that has significant influence in the Web Development community. However, Web 2.0 is still an imprecise concept because it has been defined [18] in comparison with the common practices of "Web 1.0" that were unsuccessful. From the authors' point of view, Web 2.0 can be defined from two complementary perspectives: a social one and a technological one.

From the social perspective, in "Web 1.0" websites, the end-user was a passive consumer of information that had been defined by the Webmaster. This situation has changed considerably with the arrival of Web 2.0. Today end-users not only define the website content (news, reviews, images, etc.), but they also decide which content is the most interesting. The first consequence of this role reversal has been an exponential growth of the content created by end-users on the Web.

From the technological perspective, the evolution of Web development technologies has been decisive in encouraging the end-user involvement. When the UI from the most popular Web 2.0 sites are analyzed, a high level of usability is

G. Vossen, D.D.E. Long, and J.X. Yu (Eds.): WISE 2009, LNCS 5802, pp. 131–144, 2009.
© Springer-Verlag Berlin Heidelberg 2009

detected. To achieve this improvement in usability, the evolution of Web technologies has played an essential role. This technological evolution has defined a new application paradigm called Rich Internet Applications (RIA) [5]: a Web application in which the UI is processed on the client side and the Business logic is defined by a services backend. Examples of technologies for developing RIA are Javascript frameworks, which use AJAX calls to perform functionality requests on-demand; and specific RIA platforms such as Microsoft Silverlight or Adobe Flex [1], which introduce their own languages to define complex UI. This new application paradigm has also led to new research discussions. One of the most interesting topics that have been raised is the possibility of developing RIA from a Model-driven Engineering (MDE) approach. Since several Web 2.0 applications are currently being developed using RIA technologies, providing methods, models and tools to deal with these complex technologies has become an attractive research area.

In recent years, the Web Engineering community has defined several methods with the goal of improving the development of Web applications. These methods have mainly focused on the most common domains from the early part of this decade: e-commerce and data-intensive applications. To develop "Web 1.0" interfaces, several Web Engineering methods such as WebML [5], OOHDM [23], WSDM [26], OOWS [12], and others [22] have proposed conceptual models to deal with the Web UI specification. These models have provided interesting results for defining traditional HTML interfaces in which navigation and data retrieval are the main interactions between the user and the system. However as some authors have already addressed, these methods must be extended in order to provide the new required expressivity in the RIA domain [21][7].

Even though the models proposed by these methods are essential, UI models that address more complex UI are still required. First, RIA technologies provide a wide array of UI components to define complex interaction. Also, current UI models are limited to defining the static view (layout and aesthetic properties), while Web 2.0 also requires the specification of the reactions to user events, i.e. the dynamic view. Another issue to address is the technological heterogeneity that surrounds Web 2.0 because of the different RIA technologies that are currently available. For these reasons is not a viable task to defining an all-encompassing model that covers all the available technologies.

The main contribution of this paper is to establish the foundations for supporting the UI technological perspective of the Web 2.0 in a MDE scenario. Applying the MDE principles, the technological complexity of RIA development is abstracted to the analysts. To achieve this goal the first step is the definition of a RIA Metamodel to support the new expressivity required: 1) the definition of a UI as the composition of richer widgets; and 2) the definition of the reactions of the UI as a consequence of the events triggered by end-users. The strategy for defining this metamodel is based on the analysis of different Web 2.0 applications. The most frequently used widgets and interactions (those cannot be defined using the traditional methods) have been selected. The metamodel that has been defined is generic enough to be extended and related to different MDE methods. Furthermore, it can be used as a basis to define concrete UI metamodels to address the UI modeling for different RIA technologies. To illustrate how this RIA metamodel can be applied, the integration with the OOWS Web Engineering method is also discussed.

The paper is organized as follows: Section 2 introduces the RIA Metamodel by presenting both the static view, which specifies the UI layout and composition, and the dynamic view, which defines the UI interaction. Section 3 explains the integration of the RIA Metamodel with the OOWS Web Engineering method. Section 4 presents several related works that also deal with the Model-driven development of RIA. Finally, section 5 presents conclusions and future work.

2 Defining a RIA Metamodel for Supporting Web 2.0 Applications

As stated in the introduction, there is a close relationship between Web 2.0 and the RIA domain. RIA technologies have mainly introduced changes to develop richer UI. Since the usability of the UI is a key requirement, many Web 2.0 applications have used RIA technologies to improve it. RIA technologies have introduced two main concerns that must be supported at the UI modeling level:

1. *Richer UI widgets:* The main difference between the UI produced using HTML and RIA technologies is the set of available UI widgets. Traditionally, the HTML language is constrained to a small set of elements that can be used to build an UI. This is because HTML was designed to show and to hyperlink documents rather than as an UI language. Due to the evolution of Web interfaces, HTML has become obsolete for designing UI. RIA technologies have solved this issue by migrating a lot of UI widgets (such as Datagrids or Menu Controls) from Desktop environments. RIA technologies also include specific UI widgets for multimedia visualization (images and videos) and rich editors to create the content for Web 2.0 applications.
2. *Event-driven interaction:* RIA technologies provide highly interactive UI in order to improve user experience. This interaction is triggered by the events that are performed by the end-user over the UI widgets (for instance, a mouse click). The possible responses or reactions to these events range from a change in the UI to the request for data from the Business Logic. This event-driven paradigm is not completely original because it has already been widely applied in Desktop environments. However, it has not been adopted on the Web due to the small number of events that are supported by the HTML language. RIA technologies can provide a wide set of UI events and specific programming languages to solve this issue.

Following sections present, a RIA Metamodel to support the above-mentioned concerns: the modeling of the UI as a composition of richer widgets, and the definition of the response to the UI events.

2.1 Modeling the RIA Interface

A common agreement is to define a UI as a composition of widgets: visual components of the UI to provide the data and to hold the interaction with the user [15][30]. From the analysis of several Web 2.0 applications that use RIA technologies, widgets can be categorized into five non-excluding groups according to their interactive function:

1. *Dataview widgets*: these widgets are used to show a set of structured data retrieved from the system. The *Table* widget is the most common widget used for this task. In the RIA domain, widgets of this kind of widgets have evolved considerably to show multimedia data and to provide advanced data manipulation or retrieval.
2. *Input widgets*: these widgets handle the input of the data. Depending on the analysis performed, there is a strong relationship between the type of the data and the input widget. For instance, string values are input using the *Text Input* widget, while boolean values are introduced using the *Checkbox* widget. In Web 2.0 applications like wikis or blogs, the *Rich Text Editor* widget is widely used to introduce formatted text.
3. *Navigation widgets*: these widgets are used to change the point from which the UI is perceived. The *Link* is the most common navigation widget on the Web, but RIA has introduced alternatives such as *Tab Navigations.*
4. *Service widgets:* these widgets start the execution of a service from the Business Logic. The *Button* is the default widget of this category. RIA technologies have introduced several widgets for this task such as *Icons* or *Contextual Menus*.
5. *Layout widgets*: these widgets are useful for arranging other widgets, even though they do not provide any direct interaction with the user. Layout widgets are also required to group 'child' widgets in order to perform interaction over the whole group (e.g. to hide the widgets at the same time). Examples of these widgets are *Form*, *Panel* or *Vertical Box*.

These are the most common functions but there are also widgets that match two or more categories. One example is the *Editable Datagrid,* which can be used for visualization, input of data and invocation of an update service. From the analysis performed in this study, two main ideas are pointed out for the definition of a UI using a RIA technology:

- The UI has been extended mainly by using widgets that are common in desktop environments. However, specific RIA widgets have also been provided to visualize multimedia data and to simplify the creation of content.
- The UI is highly coupled to the target RIA technology. For instance, JavaScript frameworks do not provide the same set of widgets as Adobe Flex.

The first idea implies the extension of the Web Engineering method models, using the set of widgets that is required. However, that extension should not be defined in the models that address the navigation of the Web application in order not to mix different concerns. In recent years, the HCI community has addressed UI modeling defining two levels of abstraction [4]: an Abstract level, in which the UI is defined without taking into account any technological details; and a Concrete level, in which the Abstract level is extended using the information related to the target technology. This approach can be applied in current Web Engineering methods. As a matter of fact, the navigation and presentation models proposed by the Web Engineering methods can be reused to define some of the expressivity of the Abstract UI level. This can be mainly justified since these models are not tightly coupled to a specific technology even though they are Web-oriented. Thus by including a Concrete UI level, the new technological requirements can be addressed and these models can be preserved without mixing different concerns.

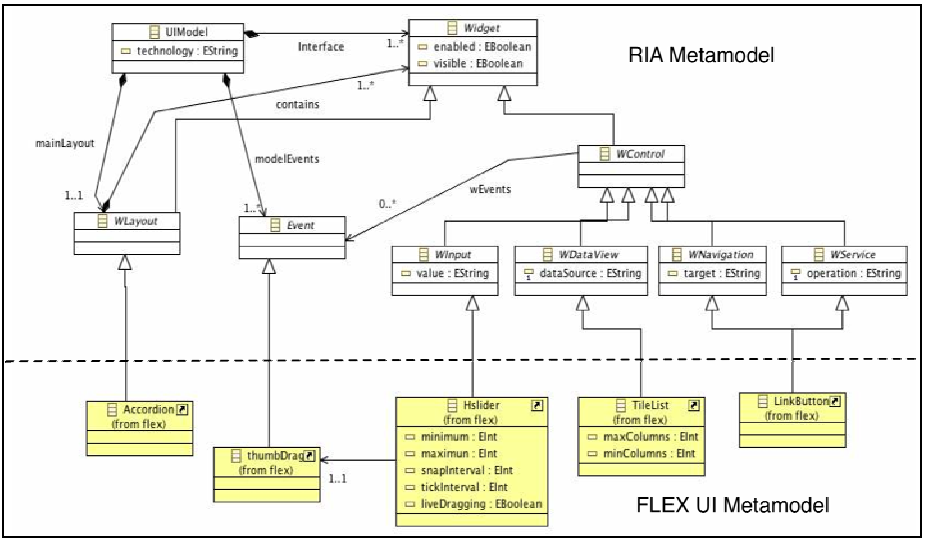

Fig. 1. The RIA Metamodel: UI view

The second idea implies the definition of a Concrete UI metamodel for each RIA technology. This set of technological-specific metamodels will define the Concrete UI models, so the advanced UI features of each technology can be introduced into the development process. Summarizing, a suitable solution requires: 1) the definition of the UI modeling at two levels of abstraction (Abstract and Concrete); and 2) the definition of one metamodel for each technology introduced into the code generation process.

Our approach proposes a RIA Metamodel to define the models of the Concrete UI level and their relationships with the models of the Abstract level (see Fig. 1). This metamodel is made up of a set of abstract modeling entities (depicted in white) from which the technological modeling entities (depicted in gray) can be defined by means of specialization relationships. This approach resembles how the UI frameworks have traditionally been developed. These abstract modeling entities are the parent entities used to create a metamodel for each specific RIA technology. Afterwards, these technology-specific metamodels are used to model the UI of the Web application for a specific RIA technology, so he analyst can define models to address all the features that a specific RIA technology can provide.

The main advantage of this approach is that the link between the Abstract level, which is made up of the Navigational Models, and Concrete level is performed using those abstract modeling entities that are common to all the technological metamodels. A suitable relationship is created between each modeling entity, which is used for defining the UI, from the Abstract level, and an abstract modeling entity from the RIA metamodel. Therefore the links between the Abstract and the Concrete levels can be reused among different technologies.

The main entity of the RIA Metamodel is a UI Model that is made up of Widgets that abstract a specific UI component. A Widget is defined by a set of Properties,

which are specified in the metamodel as meta-attributes. For instance, two properties that are common to any Widget are visible (to show or hide a specific widget in the UI) and enabled (to allow or disable the interaction with the user). The interactions that users can perform over a widget are described as Events. Widgets are divided into two types depending on whether or not they can receive Events: *Layout Widgets* (WLayout entity), which are used to arrange and to contain other widgets; and *Control Widgets* (WControl entity), which can be associated to Events in order to define reactions to the user interaction. The entity WControl is also specialized into four abstract modeling entities: WDataview, WInput, WService, and WNavigation. Each entity represents the widgets that can perform one of the specific interactive functions described above. Events are related to the UI Model entity because they are reused among the different control widgets by means of association relationships.

A technology-specific metamodel must be defined using these abstract modeling entities. For instance, if the Adobe Flex platform [1] is selected as RIA technology, a Flex UI metamodel must be defined. In order to create this new metamodel, each widget provided by Flex is defined as a child entity from one or more entities of the RIA Metamodel. Flex provides a rich UI framework that enables the implementation of complex RIA. Since this framework is built around two main concepts very close to the RIA Metamodel (interface components and containers), the specialization relationships are straightforward. The different containers are defined as specializations from the WLayout entity whereas the interface components are specialized from a suitable WControl entity depending on their function.

For example, Figure 1 (bottom) shows a brief view of the Flex UI metamodel. The *Accordion* widget is a child entity of WLayout, and *TileList* or *Hslider* are child entities of WDataview and WInput, respectively. Another example is the *LinkButton* that can be used for both navigation and service execution purposes. From these technological widget entities the Events and the Properties are created. For example, in the *Hslider* widget, the property *snapInterval* defines the increment of the slider bar, whereas the *thumbDrag* event is triggered when the user moves the slider to a new position. The final result of this process is a metamodel from which a UI for the Flex platform can be modeled.

2.2 Modeling the Event-Driven Interaction

In section 2.1, a metamodel to specify the static view of a UI has been defined. However, the UI is not only a static entity because it reacts according to the interaction with the user. This interaction is perceived as events that occur over a specific widget. This dynamic view must also be included in order to develop interactive Web 2.0 applications. To define this dynamic view, several Web 2.0 applications have been analyzed by the authors, in order to select the most common reactions to events. The selected reactions are classified as follows:

1. *Changes to the UI*: a common reaction to events is a change of the UI properties perceived by the user. For example, when the user clicks on tab a new group of widgets is shown.
2. *Request for data on-demand*: the result of this reaction is a request for information from the server. Usually, only the data that has not been previously retrieved and stored is requested.

3. *Functionality execution*: this reaction implies a request-response communication with the Business Logic. When the response is received, a feedback message is presented to the user to inform about the result.
4. *Input Validation*: this reaction informs the user about mistakes and suggests a solution. When the end-user introduces a value, mistakes are common: from syntactic errors (e.g. a date with an incorrect format), to more complex ones that involves the Business Logic (e.g. to check whether or not the login provided has already been created).
5. *Navigation:* A navigation changes the current point from which the application UI is perceived by the user and the interactions available. RIA has included animation effects to highlight this reaction.

The event rule concept is introduced to describe this dynamic view made up of reactions. An event rule is defined by an Event from a source widget that triggers a Reaction associated with a target widget. This approach resembles the definition of *Event-Condition-Action* (ECA) rules proposed by the HCI community to perform the same goal [14]. The modeling entities to define the event rules are defined in the RIA Metamodel as Figure 2 illustrates.

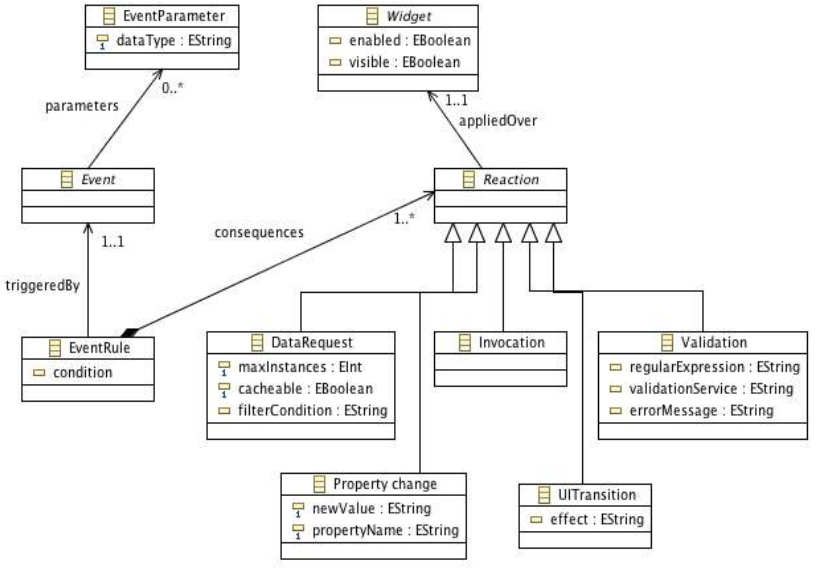

Fig. 2. The RIA Metamodel: Dynamic View

Event rules are defined over the widgets from a technological UI model specialized from the RIA Metamodel. The scope of the event rules is the layout container where are defined the widget that triggers the event and the widget that receives the reaction. Hence, an event rule can generate a reaction to several widgets as long as they are defined in the same parent layout. Additionally, the widget that produces the event and the target widget that performs the reaction can be the same. Optionally, events

can include parameters with additional information to configure the reaction. Each event rule is also associated to at least one reaction of the five described above. Each reaction is represented by a modeling entity in the RIA metamodel, and it is also linked to the specific type of widget that supports the expected behavior. These five reaction entities are described below:

- *Property change:* this entity represents a change of the current value of any property of the target widget. By applying this reaction the state of a widget can be controlled according to the triggered events. Since all widgets are described using properties, this reaction can be associated to any widget.
- *Data Request*: this entity represent a request of data from the Business Logic in order to populate a Dataview widget. This reaction can be parameterized by means of the following attributes: the cacheable attribute, which provides a mechanism to avoid having to request previously retrieved data; the max instances attribute, which defines the maximum number of instances to be retrieved; and the filter condition attribute, which selects the data depending on a logic formula.
- *Invocation*: this entity represents an asynchronous service execution from the Business Logic. The user can interact with the rest of the UI while the service is being executed. This reaction is always linked to a Service widget. Moreover, this reaction triggers by default an event to inform whether or not the service has been successfully executed. That event can be used to trigger new event rules that take into account the service results.
- *Validation*: this reaction verifies whether or not the data introduced by the user is correct. The validation can be defined using a regular expression to be satisfied by the input or a Boolean service that receives the input as an argument. If the regular expression or the service result is evaluated as false, an error message is shown to the user.
- *UI Transition*: this reaction represents a transition that is associated with a Navigation widget. Additionally, a visual effect to highlight the transition can be defined.

As an example to illustrate how an event rule can be defined the suggestion pattern is used: when inputting new data, (event) the client-side automatically retrieves (without refreshing the UI) the values that match the input text. In this pattern, two Flex widgets are required to define the UI: a Text Input, and a List widget to show the suggestions. A Data request entity is associated to the List widget and the filter condition of this entity is changed as the user inputs a new text value. This event rule is defined below using a textual notation (for exemplification purposes) that conforms to the metamodel entities presented:

```
SET  RequestReacton TO List:
     RequestReaction.cacheable = true;
ON:
     TextInput.dataChange(newValue)
DO:
     RequestReaction.filterCondition = "LIKE " + newValue;
     List.Request();
```

3 RIA Modeling in the OOWS Web Engineering Method

The RIA metamodel proposed in Section 2 has been defined without taking into account a specific method. Therefore, the main issue is how the new RIA models are introduced and linked with the previous models of the method. This section presents a strategy to deal with this integration. As a proof of concept, this strategy has been applied to the OOWS Web Engineering method [12].

OO-Method [19] is an automatic code generation method that produces an equivalent software product from a system conceptual specification. OOWS was defined to extend OO-Method with the principles proposed by the Web Engineering community. To achieve this goal, OOWS introduces a new set of models for supporting the interaction concern between the user and a Web application (See Figure 3). These models are:

- *User Model*: This model is a user diagram to specify the types of users that can interact with the system.
- *Navigational Model:* This model defines the system navigational structure for each type of user by means of a Navigational Map. This map is depicted by means of a directed graph whose nodes represent Navigational Contexts and whose arcs represent navigational links that define the valid navigational paths. The different Navigational Contexts are described as an aggregation of one or more Abstract Interaction Patterns (AIP). An AIP defines the interaction in terms of two main communication flows between the user and the system: a) a population retrieval flow, which provides information from the system to the user; and b) a service execution flow started by the user, which performs a change of the state of the Information System. Two main AIP, Population and Service, are defined. These AIP are linked to the data and functionality represented by the OO-Method models. In order to constrain the behavior and/or to refine these two main interactions more accurately, auxiliary interaction mechanisms are also introduced.

Currently OOWS generates the Web Application code from these abstract models Further details about the OOWS models and the code generation process can be consulted in [28] and [29] respectively. In order to extend OOWS with the RIA Metamodel, the modeling phase is divided in two levels: an Abstract level made up of the current OOWS models and a new Concrete UI level. This new level adds a new methodological step that is highlighted in Fig. 3. In this new step (Concrete UI modeling), the analyst can choose between the modeling of a traditional HTML interface or a Flex UI. The following subsection explains how the link between the models of the two levels is established.

3.1 Integrating the RIA Metamodel with the OOWS Method

Current abstract models proposed by Web Engineering methods provide information that must be taken into account when the final UI code is generated. For instance, these models provide the data structures that must be shown using a Dataview widget. Since in the approach presented, the UI is defined using the RIA Metamodel, the integration with the previous OOWS models must also be addressed.

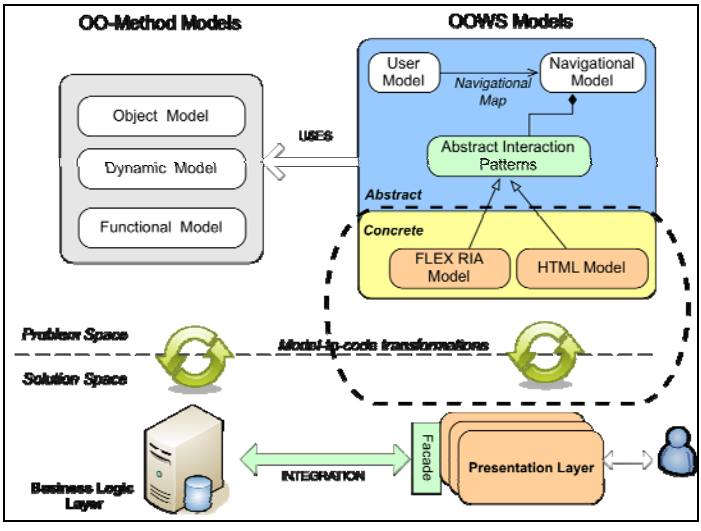

Fig. 3. The extended OOWS MDE development process

A common strategy in a MDE environment is to define a model-to-model (M2M) transformation from an abstract model to a more technological one. Hence, the technological model replaces the previous one and provides the additional expressivity required. However, the use of M2M transformations has two main disadvantages: 1) since the technological models must include all the expressivity from the abstract model they are complex to define; and, 2) since in practice, each abstract modeling entity can be represented in the UI using different RIA widgets, the number of M2M transformations to be defined is high.

To overcome these issues, the solution proposed in this work is to create relationships instead of transformations between the method metamodel and the RIA metamodel. To capture relationships of this type, weaving models have been applied in other works [6]. A weaving model [11] establishes the relationship between the modeling entities of two models that are not directly related. Weaving models are defined conforming to a weaving metamodel that defines the two metamodels to be connected and the set of relationships between the modeling entities. Therefore, neither of the two metamodels is modified because all the knowledge about the relationships between them is defined in the weaving metamodel.

For this work, a weaving metamodel has been defined between the RIA metamodel and the OOWS method metamodel. First the two metamodels were defined using the Eclipse Modelling Framework [3] because Eclipse tools such as the Atlas Model Weaver [8] and the Epsilon plug-in [10] can be used to implement the weaving metamodel. Figure 4 shows a set of relationships defined in the OOWS weaving metamodel. The main function of these relationships is to establish which widget type is going to represent an OOWS modeling entity. For instance, in OOWS the Population AIP defines the set of data to be shown to the user in a web page. Each Population AIP defined in an OOWS model must be related to a Dataview widget. When the analyst creates the weaving model, a Flex widget, which is specialized from

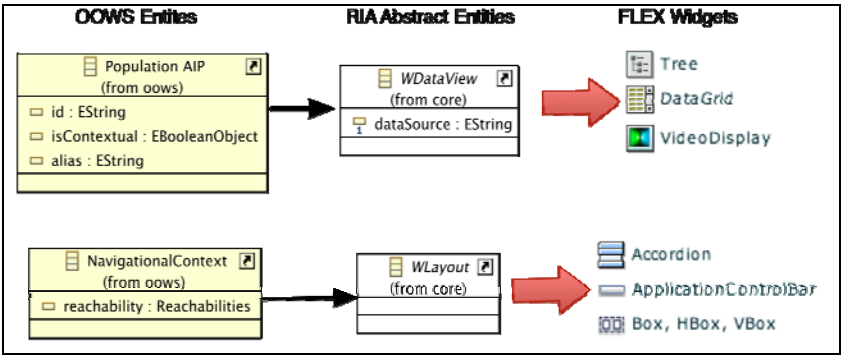

Fig. 4. Weaving relationships between the OOWS Metamodel and the RIA Metamodel

the DataView entity, must be selected for each Population AIP. The same line of reasoning is applied to the rest of the OOWS modeling entities that must be reflected in the UI.

The main advantage of this approach is that the weaving metamodel is defined between the specific method and the abstract entities of the RIA metamodel. Therefore, if another RIA technology is selected, the weaving metamodel does not need to be modified because the relationships are established with the abstract entities instead of the Flex technological entities. Furthermore, it is easier to create and maintain several weaving models than different sets of M2M transformation rules.

Once the analyst has defined the RIA model, event rules are specified using the dynamic view of the metamodel presented in section 2. In order to support the definition of these rules in the OOWS method, the XText framework has been used [17]. This framework creates an Eclipse metamodel to support the definition of a given textual grammar. Therefore, that metamodel has been included in the RIA Metamodel and related with the correspondent Event and Reaction entities.

Finally, the OOWS model compiler must be redefined because the new input models are: 1) an OOWS model that describes the interaction of the user with the system; 2) a RIA model that specifies the UI and the event rules; and, 3) a weaving model that establishes the relationships between these two models. Taking these three models as input, the final code (for instance, a Flex UI) is generated by means of model-to-code transformations. The specification of this code generation process is out of the scope of this paper.

4 Related Work

Several works in the Web Engineering field have proposed methodological extensions to support RIA development. First, Bozzon and Comai [5] have extended the WebML method to support RIA modeling. The proposed extension determines the definition at the modeling level which data, operations, hypertext links, and page contents must be processed by the client side. Additionally, this work proposes a code generation process to obtain the final application using a RIA technology. In the context of the OOHDM method, Urbieta et al. [27] propose an approach for designing RIA

interfaces, taking into account the separation of the interface structure from the behavior. That work proposes an aspect-oriented perspective to combine different concerns related to the UI composition. Hence, the UI is defined as a composition of several interface atoms. They also propose [24] a set of refactorings to transform conventional Web interfaces to RIAs.

The OOH method has also been extended defining the OOH4RIA approach [15] in which two models are defined: a Presentation Model and an Orchestration Model. The Presentation Model has been defined using the Google Web Toolkit (GWT) as the technological framework to define the metamodel entities. The Orchestration Model [20] defines the interaction dependencies between widgets and how these widgets should be grouped to improve the requests to the server. The need for better interaction models has also been pointed out by Dolog and Stage [9]. They propose the use of UML statechart diagrams to support the synchronization between the client side and the server side of a RIA as well as between widgets.

The common agreement in the works mentioned above is that the current Web Engineering methods must be extended at the conceptual level in order to deal with Web 2.0 development. Our work proposes a generic RIA metamodel that has been specifically defined to be method-independent. This metamodel combines both the static and dynamic view of the RIA UI that previous works have addressed using different method-specific models. Furthermore, the previous works have mainly focus on how to model the logic and data persistence on the client side of a RIA application. Even though this issue is relevant, in our view the main advantage of RIA is the improvement in the UI usability. Our approach also considers that the methodological extension required cannot be achieved only by extending the current method models. A new methodological phase must be included in order to deal with the Concrete UI specification.

Furthermore, several works have addressed the RIA development from a HCI perspective. A metamodel is proposed by Ruiz [25] for defining a RIA UI from an Abstract Interface Model. In that work, a clear relationship is established between the abstract and the concrete UI that represents the interaction. Another interesting approach is the RUX-method [13]. This method proposes how to define at the concrete level the time-related behaviors (Temporal Presentation) and the event-response actions (Interaction Presentation) in order to define RIA interface components. The main difference between our proposal and these works is the expressivity provided by the concrete UI level. Their concrete models are defined with a set of widgets that are common to different RIA technologies. As a consequence, is not possible to take full advantage of the advanced features of a specific RIA technology.

5 Concluding Remarks

In this paper, a model-driven UI development approach for Web 2.0 has been presented. Because RIA technologies are very common in the Web 2.0 development, a metamodel for dealing with the UI of this application paradigm has been proposed. Several lessons have been learned during the development of this approach: The first lesson is that Web Engineering methods provide suitable models that can also be reused for Web 2.0 development. However, problems arise when specific technological details must be introduced. Therefore, a UI model-driven development

process for Web 2.0 must consider the target technology. The second lesson is that the clear separation of abstraction levels proposed by HCI approaches can be easily adopted. Finally, the third lesson is that defining a single UI model to support every technology in Web 2.0 is an unfeasible task. The approaches to support RIA development must be flexible enough to be easily adapted to different technologies.

It should be pointed out that the manual definition of the weaving model might be a tedious and error-prone task, if the number of entities in the model is very large. A potential solution to this issue is the use of patterns. These patterns can abstract several best practices in an understandable way and be reused to avoid the full definition of the weaving model.

Further work will address full tool support of the Web 2.0 development process presented in this work. Currently, the RIA Metamodel and several model-to-code transformation rules for the Adobe Flex platform have been defined for validation purposes. A Web 2.0 application for research management is being developed using the ideas presented in this paper. However, the final goal is to apply these ideas in an industrial tool. Finally, integration with other Web Engineering methods besides OOWS must be analyzed in order to validate the flexibility of the RIA Metamodel.

Acknowledgments. This research work has been developed with the Spanish MEC support under the project SESAMO TIN2007-62894 and the FPU grant AP2005-1590.

References

1. Adobe Flex Developer Center, `http://www.adobe.com/devnet/flex/` (accessed April 2009)
2. Bozzon, A., Comai, S.: Conceptual Modeling and Code Generation for Rich Internet Applications. In: 6th International Conference on Web Engineering (ICWE), California, United States (2006)
3. Budinsky, F., Merks, E., Steinberg, D., Ellersick, R., Grose, T.J.: Eclipse Modeling Framework. Addison-Wesley Professional, Reading (2003)
4. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A Unifying Reference Framework for multi-target user interfaces. Interacting with Computers 15, 289–308 (2003)
5. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. Morgan Kaufmann, San Francisco (2003)
6. Cetina, C., Fons, J., Pelechano, V.: Applying Software Product Lines to Build Autonomic Pervasive Systems. In: Proceedings of the 2008 12th International Software Product Line Conference. IEEE Computer Society, Los Alamitos (2008)
7. Comai, S., Carughi, G.T.: A Behavioral Model for Rich Internet Applications. In: 7th International Conference in Web Engineering, Como, Italy (2007)
8. Del Fabro, M., Bézivin, J., Valduriez, P.: Weaving Models with the Eclipse AMW plugin. In: Eclipse Summit Europe, Esslingen, Germany (2006)
9. Dolog, P., Stage, J.: Designing Interaction Spaces for Rich Internet Applications with UML. In: Baresi, L., Fraternali, P., Houben, G.-J. (eds.) ICWE 2007. LNCS, vol. 4607, pp. 358–363. Springer, Heidelberg (2007)
10. Epsilon, `http://www.eclipse.org/gmt/epsilon/` (accessed May 2009)
11. Fabro, M.D.D., Valduriez, P.: Semi-automatic model integration using matching transformations and weaving models. In: Proceedings of the 2007 ACM symposium on Applied computing. ACM, Seoul (2007)

12. Fons, J., Pelechano, V., Albert, M., Pastor, O.: Development of Web Applications from Web Enhanced Conceptual Schemas. In: Song, I.-Y., Liddle, S.W., Ling, T.-W., Scheuermann, P. (eds.) ER 2003. LNCS, vol. 2813, pp. 232–245. Springer, Heidelberg (2003)
13. Linaje, M., Preciado, J.C., Sánchez-Figueroa, F.: Engineering Rich Internet Application User Interfaces over Legacy Web Models. IEEE Internet Computing, 53–59 (2007)
14. Mbaki, E., Vanderdonckt, J., Guerrero, J., Winckler, M.: Multi-level Dialog Modeling in Highly Interactive Web Interfaces. In: 7th International Workshop on Web-Oriented Software Technologies, vol. 445, pp. 38–43. WS-CEUR, New York (2008)
15. Meliá, S., Gómez, J., Pérez, S., Díaz, O.: A Model-Driven Development for GWT-Based Rich Internet Applications with OOH4RIA. In: Eight International Conference on Web Engineering. IEEE Computer Society, New York (2008)
16. Noda, T., Helwig, S.: Rich Internet Applications - Technical Comparison and Case Studies of AJAX, Flash, and Java based RIA. In: Best Practice Reports University of Wisconsin-Madison, vol. 2008 (2005)
17. OpenArchitectureWare. Xtext Reference Documentation, `http://wiki.eclipse.org/Xtext/Documentation` (accessed April 2009)
18. Oreilly, T.: What is Web 2.0? Design Patterns and Business Models for the Next Generation of Software (2005), `http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html` (accessed April 2009)
19. Pastor, O., Molina, J.C.: Model-Driven Architecture in Practice. In: A Software Production Environment Based on Conceptual Modelling. Springer, Heidelberg (2007)
20. Pérez, S., Díaz, O., Meliá, S., Gómez, J.: Facing Interaction-Rich RIAs: the Orchestration Model. In: Eight International Conference on Web Engineering, pp. 24–37. IEEE Computer Society, New York (2008)
21. Preciado, J.C., Linaje, M., Sánchez, F., Comai, S.: Necessity of methodologies to model Rich Internet Applications. In: 7th IEEE International Symposium on Web Site Evolution, pp. 7–13. IEEE, Budapest (2005)
22. Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.): Web Engineering: Modelling and Implementing Web Applications. Springer, Heidelberg (2008)
23. Rossi, G., Schwabe, D., Lyardet, F.: User interface patterns for hypermedia applications. In: Proceedings of the working conference on Advanced visual interfaces, pp. 136–142. ACM Press, Palermo (2000)
24. Rossi, G., Urbieta, M., Ginzburg, J., Distante, D., Garrido, A.: Refactoring to Rich Internet Applications: A Model-Driven Approach. In: Eight International Conference on Web Engineering, pp. 1–12. IEEE Computer Society, New York (2008)
25. Ruiz, F.J.M.: A Development Method for User Interfaces of Rich Internet Applications. DEA Thesis. Université catholique de Louvain, Louvain, Belgium (2007)
26. Troyer, O., Casteleyn, S., Plessers, P.: WSDM: Web Semantics Design Method. In: Web Engineering: Modelling and Implementing Web Applications, pp. 303–352. Springer, Heidelberg (2008)
27. Urbieta, M., Rossi, G., Ginzburg, J., Schwabe, D.: Designing the Interface of Rich Internet Applications. In: Fifth Latin American Web Congress (LA-WEB), Santiago de Chile (2007)
28. Valverde, F., Panach, I., Aquino, N., Pastor, O.: Dealing with Abstract Interaction Modelling in an MDE Development Process: a Pattern-based Approach. In: New Trends on Human-Computer Interaction, pp. 119–128. Springer, London (2009)
29. Valverde, F., Valderas, P., Fons, J., Pastor, O.: A MDA-Based Environment for Web Applications Development: From Conceptual Models to Code. In: 6th International Workshop on Web-Oriented Software Technologies, Como, Italy, pp. 164–178 (2007)
30. Vanderdonckt, J., Limbourg, Q., Michotte, B., Bouillon, L., Trevisan, D., Florins, M.: USIXML: a User Interface Description Language for Specifying Multimodal User Interfaces. In: WMI 2004, Sophia Antipolis, Greece (2004)