A Model-Driven Development for GWT-Based Rich Internet Applications with OOH4RIA

Santiago Meliá, Jaime Gómez
IWAD Group, University of Alicante
Alicante, Spain
{santi,jgomez}@dlsi.ua.es

Sandy Pérez, Oscar Díaz
ONEKIN Group, University of Basque Country
San Sebastián, Spain
sandy-perez@ikasle.ehu.es, oscar.diaz@ehu.es

Abstract

Traditionally, Web applications have had great limitations in the usability and interactivity of their user interfaces. To overcome these limitations, a new type of Web applications called Rich Internet Applications (RIAs) has recently appeared providing richer and more efficient graphical components similar to desktop applications. However, RIAs are rather complex and their development requires the designing and implementation tasks which are time-consuming and error-prone. Moreover, RIA development is a new challenge for the Web engineering methodologies requiring their modification and the introduction of other concerns. In this context, we propose a new approach called OOH4RIA which proposes a model-driven development process that extends OOH methodology. It introduces new structural and behavioural models in order to represent a complete RIA and to apply transformations that reduce the effort and accelerate its development. This RIA will be implemented on the promising Google Web Toolkit (GWT) framework.

1. Introduction

Traditional Web applications developers have focused all their activity around a client-server architecture where all processing is done on the server side and a thin client which is only used to display static contents. This approach has suffered significant drawbacks and limitations, especially due to the richness of the application interfaces and the overall sophistication of the solutions that could be built and delivered.

These old-fashioned Web applications are being replaced by the so-called Rich Internet Applications (RIAs) [3] which provide richer and more interactive user interfaces, similar to desktop applications. Moreover, RIAs provide a new client-server architecture that reduces significantly network traffic using more intelligent asynchronous requests that send only small blocks of data.

However, RIAs are complex applications and their development requires designing and implementation which are time-consuming and error-prone. In fact, the technological advances of RIAs require from the developer (1) to represent a rich user interface based on the composition of Graphical User Interface (GUI) widgets, (2) to define an event-based choreography between these widgets and (3) to establish a fine-grained communication between the client and the server layers.

The Web engineering community is well-aware that the RIA development is a new and difficult challenge that requires modifying the traditional methodologies. On the one hand, it must introduce new models in order to represent the interactive user interface and on the other hand, it needs to improve the development process using automation techniques that accelerate it and reduce errors.

In this context, the paper presents a new approach called OOH4RIA based on the MDE (Model Driven Engineering) paradigm [8] that proposes a complete development process based on a set of models and transformations allowing to obtain the implementation of Rich Internet Applications. The process uses the well-known Web design method called OOH [5] that defines the domain and navigation models allowing us to generate a CRUD server side of the RIA (defined by [5]). Both OOH models are the starting point of model-to-model transformations which establish the skeleton of the presentation and orchestration models that represent the client side of RIA. They are platform-specific because they contain widgets and properties of one of the most promising RIA frameworks: Google Web Toolkit (GWT) [6]. GWT is an AJAX framework, developed by Google, which permits us to create RIAs by writing the browser-side code in Java, thus gaining all the advantages of Java (e.g. compiling, debugging, etc.) and generating a generic Javascript and HTML code that can be executed in any browser. Moreover, GWT makes every attempt to be flexible allowing us to integrate with other client AJAX frameworks (e.g. Script.aculo.us, Dojo, Yahoo! UI, and so on) and with server Java frameworks such as Struts, EJB, etc.

978-0-7695-3261-5/08 $25.00 © 2008 IEEE
DOI 10.1109/ICWE.2008.36

In order to understand this approach, this paper presents a new model-driven development process applied to a case study called GWT Mail Application [7]. In section 2, we give an overview of the process using the model-driven specific SPEM notation. The subsequent sections detail the different artefacts of this process. Section 3 presents the OOH domain and navigation models that specify the RIA server side. In sections 4 and 5, we introduce the main contribution of this work namely the presentation and orchestration models. Once all models are defined, we propose the specification of the transformation models used for accelerating this process, as we can see in section 6. Finally, sections 7 and 8 outline the relevant related work and the future lines of research.

2. The Model-Driven development process of OOH4RIA

We propose a model-driven development process allowing to specify an almost complete Rich Internet Application (RIA), through the extension of the traditional Web methodologies like OOH with two new RIA presentation models which will be introduced in this paper.

Figure 1 is a graphical representation of the model-driven development process with definition of models and transformations that permit to obtain a RIA implementation. A notation proposed by the OMG standard SPEM represents this process. This diagram introduces a set of stereotypes specific for model-driven processes. It includes an actor stereotype able to represent a transformation engine called *Model Transformer* and defines a set of stereotypes of the metaclass activity to represent different MDA transformations such as PIMToPIM, PIMToPSM, PIMToCode, PSMToCode, etc.

The process starts with the *OOH designer* defining the OOH domain model to represent the domain entities and the relationships between them. From this model, the *OOH designer* represents the navigation through the domain concepts and establishes the visualization constraints using the navigation model. With the definition of both OOH models begins the part of the process that constitutes the main contribution of this work.

It first transforms the navigation model into the presentation model by means of the PIM2PSM transformation called *Nav2Pres*. This presentation model is specifically defined for the GWT platform allowing us to capture clearly the different widgets that constitute a GWT interface. The Nav2Pres is a model-to-model transformation defined in QVT that establishes the different screenshots of the model presentation.

After obtaining the container screenshots of the presentation model, the User Interface designer completes them placing the widgets, defining the style and establishing the spatial configuration by means of Panels. It is worth pointing out that these widgets can be related to a navigational element thereby showing the dynamic content coming from the server side into the user interface.

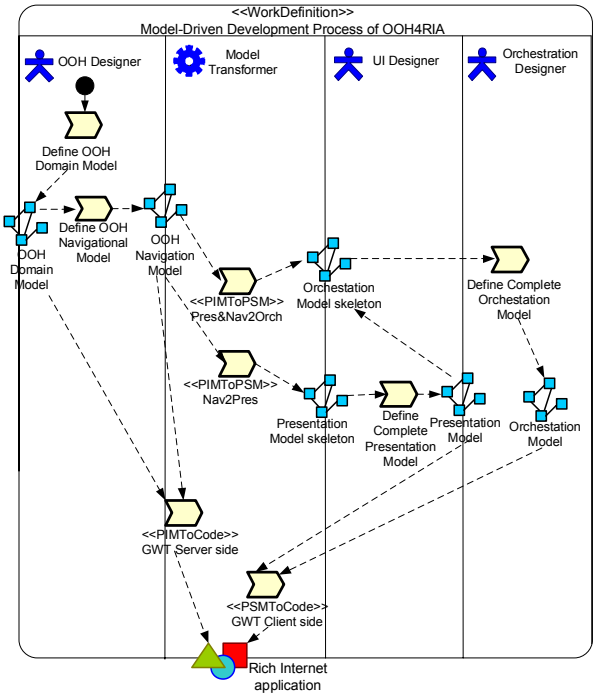

Figure 1. The OOH4RIA model-driven development process

Since the RIA possesses a rich interactive user interface similar to desktop applications, we must complete the static features of widgets with a model that will allow us to specify the interaction between these widgets and the rest of the system. This model has been called orchestration model and is represented as a UML profile of state machine diagram. The orchestration model does not have to be defined from scratch because a model-to-model transformation called *Pres&Nav2Orch* allows us to obtain the skeleton of the orchestration model from the navigation and presentation models.

The model-to-model transformation starts by establishing the screenshots behavioural states and their transitions from the navigational nodes and the associations respectively. It also defines the widget behavioural states corresponding to the widgets represented in the presentation model. At this point, the

designer completes the orchestration model introducing the events, operations and triggers of different states.

The last step consists in defining the model-to-text transformations that will grant us the RIA implementation. The *GWT Server Side* transformation generates the server code from the OOH domain and the navigation models, while the *GWT client side* transformation generates the client side code using a specific GWT framework. Both transformations are written in the MOFScript language which follows the OMG ModelToText RFP for the representation of model-to-text transformations.

In the following sections we present the different artefacts generated during this process applying them to a clear and simple case study: the GWT Mail application [7]. In essence, this case study demonstrates how to construct a relatively complex user interface, similar to many common email applications, and how a traditional Web methodology like OOH must be extended in order to support the representation of a richer and more interactive Web User Interface

3. The RIA server with OOH

The OOH (Object-Oriented Hypermedia) method [5] is based on the object-oriented paradigm which provides the designer with the semantics and notations necessary for the development of the traditional Web applications. The OOH defines a set of models: (1) the domain model, (2) the navigation model, and (3) the presentation model. The latter one is defined by a set of HTML elements and is represented in a non-visual XML notation. This model does not satisfy the graphical and interactive requirements of the RIA applications. Therefore, in this paper, we define a more complete presentation that aims at solving this problem. This section is focused on the domain and navigation models that give the possibility to represent the functional concepts of the GWT Mail server side.

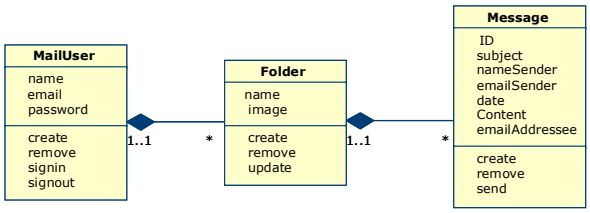

Figure 2. Domain model of the GWT Mail application

Figure 2 depicts the domain model for the Mail application example. As you can see, this model represents the most important domain entities, free from any technical or implementation details, thus representing an ideal class model. The *MailUser* accesses the application in order to send and receive different mails. He stores the messages in one or more elements of the *Folder* allowing him to classify the mails (Messages) according to his own criteria. Finally, in *Message*, the *MailUser* can find all the information about an e-mail, namely attributes such as *subject*, *nameSender*, *emailSender*, *date*, etc. as well as the operations that allow him to *create*, *remove* and *send* a Mail.

After specifying the domain model, the designer must design the navigation model which defines the navigation and visualization constraints. The navigation model is formalized by a MOF metamodel which establishes a set of different types of navigational elements. It is important to point out that the navigation model establishes the most relevant semantic paths through the information space filtering the domain elements that can be seen in the client side.

When defining the navigation model, the designer must take into account some orthogonal aspects such as the desired navigation behaviour, the object population selection, and the order in which objects should be navigated, or the cardinality of the access. Furthermore, this navigation model indicates whether the information is shown in a different Web page called “target link” (if the link arrow has a filled arrowhead) or in the same page called “origin link” (if the link arrow has an empty arrowhead). Thus, it allows us to infer the set of the screenshots that have the RIA application. These features are captured by means of different navigation elements defined by the navigation metamodel (it can be seen [2]).

Figure 3 depicts the navigation model for the GWT Mail case study. The navigation starts with the *Login* NavigationalClass where a MailUser has to identify the system in order to obtain access. The *signIn* operation activates a ServiceAssociation also called *signInLink*. If the user has introduced a valid email and password, the system navigates to the *User* NavigationalClass. If the *signIn* operation is incorrect, the application goes back to the same page. The main page is made up of the navigationalClass called *MailReader* containing the name and email of the MailUser and shows him automatically a set of his own Folders (*SelectFolder*).

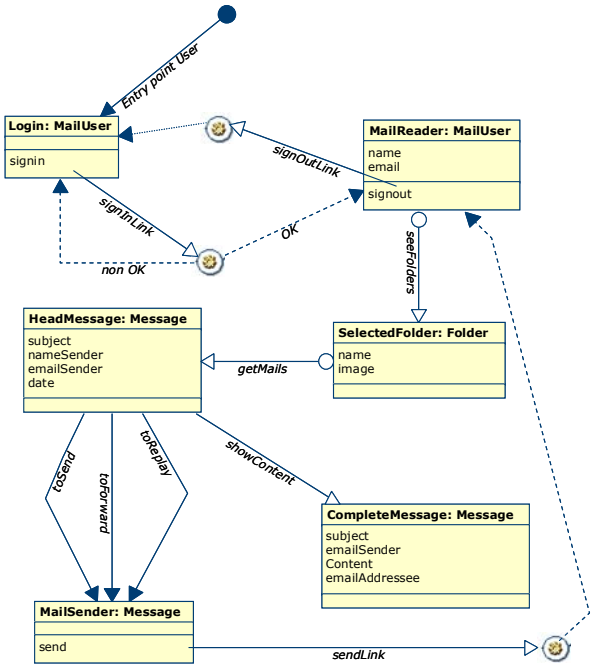

Figure 3. Navigation model of the GWT Mail application

He must now select a specific folder in order to see the messages presented by the *HeadMessage* NavigationalClass. By selecting a *HeadMessage* he will see the *CompleteMessage*. Also, he can navigate from the *HeadMessage* to the *MailSender* using some of the links (such as *toSend*, *toReply*, *ToForward*) in order to create a new message. Finally, this message is sent by the user using the ServiceAssociation called *sendLink* and the application comes back to the main page.

The next step in this process is to execute the Nav2Pres transformation. This model-to-model transformation permits us to obtain a skeleton of the presentation model creating the different container screenshots where the UI designer must represent the different RIA widgets. The Nav2Pres transformation is detailed in section 6.

The next section illustrates how to design the static features of a RIA user interface by means of the presentation model

4. The presentation model

The presentation model proposes a structural representation of the different GUI widgets that constitute the complete layout features of a RIA user interface. The presentation is defined by a specific-domain model mainly focused on obtaining similarity with the look and feel of a RIA application, thus allowing developers with a low formation in programming to specify a complete RIA interface. At the same time, this model keeps the model-driven approach being formalized by a MOF metamodel that represents the topology of GUI widgets, their properties and constraints. Using a MOF metamodel offers us three important advantages: (1) it establishes the relationships with other concerns allowing to associate GUI widgets with choreography and navigational functionality in order to establish a complete communication between the client and server layers and (2) it allows us to establish model transformations thus reducing the modelling and implementation tasks. (3) Finally, it also permits us to use frameworks like GMF to realize a graphical representation of this model in a CASE tool in a short period of time. For the sake of clarity, Figure 4 represents a simplified MOF presentation metamodel that shows the most important elements of Presentation model.

Unlike the traditional Web applications, a RIA usually follows the “simple page application” [12] where only a page with a set of stateful widgets constitutes the user interface. However, a RIA application could be very complex and these widgets are not shown at the same time and with the same appearance. Thus, a RIA user interface could be split into different screenshots containing only the widgets rendered at that moment. Following this assumption, a PresentationModel is made up of a set of ScreenShot elements.

A ScreenShot is used like a container that allows the UI designer to realize a spatial distribution of different Widget elements rendered at a given moment. The Widget is the central element of the presentation model. It permits to build a graphical user interface by means of the composition, reusability and extensibility of these components. The Widget establishes a relationship with a NavigationalElement providing dynamic information from the server side (e.g. NavigationalClass, NavigationalAttribute, etc.). Also, a Widget has an isCoordinable attribute indicating if the widget has a relevant behaviour (isCoordinable with true value) and it will be represented in the orchestration model.

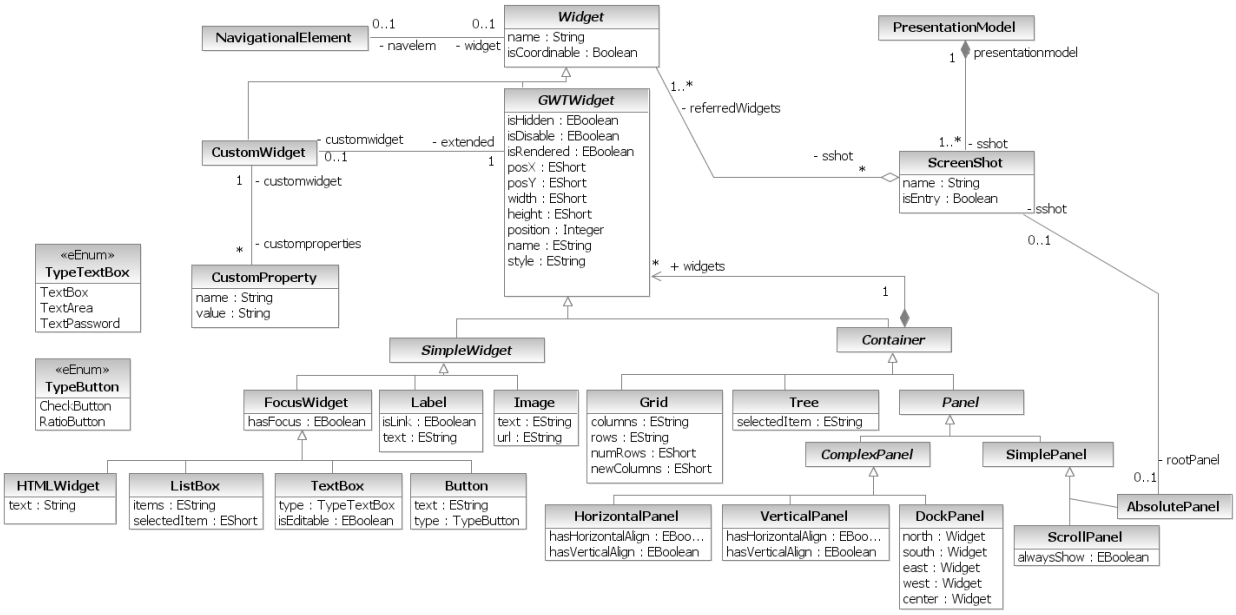

Figure 4. Simplified presentation metamodel

A widget is an abstract metaclass that could be specialized in the GWTWidget representing a concrete widget of GWT framework (e.g. TextBox, ListBox, Image, Button, Tree, etc.) or a CustomWidget. There are two subtypes of the GWTWidget: (1) the SimpleWidget which is a generic simple UI component that adds support for sending input and/or receiving output data (e.g. Button, TextBox, Label, Image, etc.). (2) A container that represents a composite component which by applying the Composite Pattern [4] can contain other widgets (simple or composite). A container can specialize in Grid and Tree widgets which propose a predefined spatial distribution and functionality. Alternatively, a container can specialize in Panel which is a generic widget that allows the UI designer to define a personalized spatial distribution of the widgets it contains. A Panel can specialize in a SimplePanel when it only contains one widget (e.g. AbsolutePanel) or a ComplexPanel when it permits to contain a set of widgets (e.g. HorizontalPanel, VerticalPanel, etc.). There is a greater variety of type of panels but for the sake of clarity the paper will only present the most relevant ones.

In order to avoid the possible limitations of widgets provided by the GWT framework, this metamodel incorporates an UI extension mechanism using the component called CustomWidget allowing the UI designer to personalize a GWTWidget introducing new graphical and functional characteristics.

Once the presentation elements have been formalized by the presentation metamodel, we are going to apply these concepts to the case study GWT Mail application. Figure 5 depicts the main ScreenShot of Mail application allowing to visualize the different mails of the user. The spatial configuration of this state is made up a root panel of the DockPanel type that divides it into three parts: the NORTH side that has the heading, the WEST side that has the menu and the CENTER side that has the mails. Starting from the heading, it locates its widgets horizontally with a HorizontalPanel which is associated with the User NavigationalClass in order to show the name and email user using two Label widgets. Furthermore, the heading also contains a Hiperlink widget called signout that sends an event to exit the application.

The menu is made up of an HTMLWidget containing a heading with an Image and a Label carrying the name of the application. At the bottom part, it has a Tree widget that contains a root element represented by a TreeItem showing an image and the email of the MailUser. This Tree also has a TreeItem associated with the Folder NavigationalClass representing a set of TreeItems with the image and name of each Folder of a MailUser.

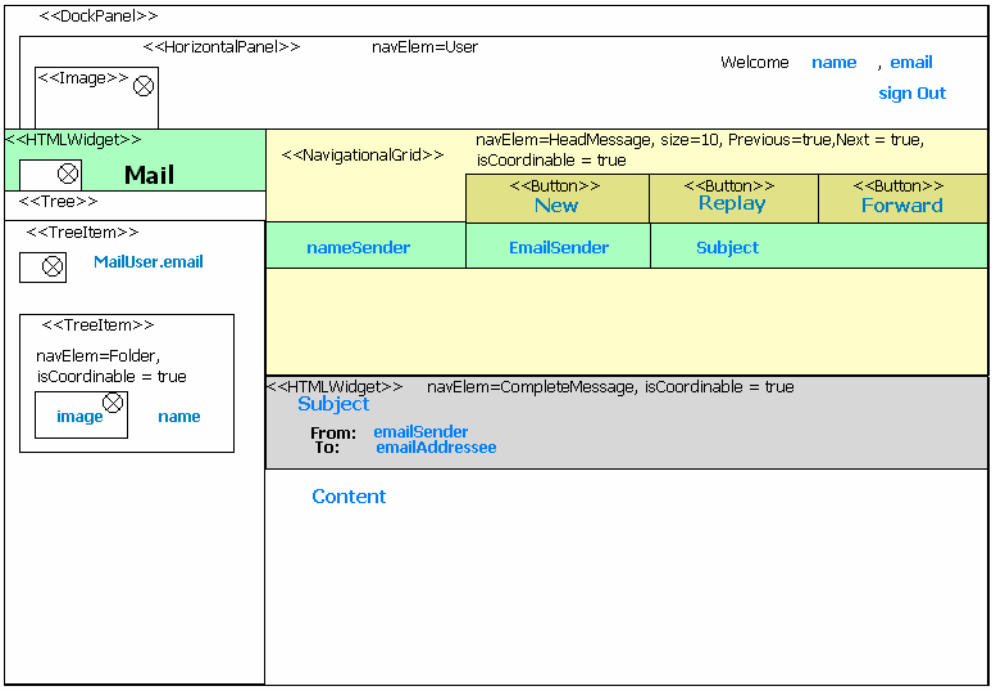

Figure 5. Presentation model of the GWT Mail application

The CENTER part contains the widgets that display the mails. On the top side, a CustomWidget called NavigationalGrid shows the message headings using a paging mechanism. To do that, we have extended the Grid widget to a CustomWidget called NavigationalGrid which has a set of properties able to realize the paging such as size, previous and next. Moreover, the NavigationalGrid is associated with the HeadMessage NavigationalClass obtaining the messages, viewing the fields for each headmessage (such as nameSender, emailSender and Subject) and providing a set of buttons (Reply, Forward and New) to create new messages. On the botton side, there are two HTMLWidget elements that show all the contents of a selected e-mail associated with a CompleteMessage NavigationalClass, the widget above shows the heading of the selected message and the widget below shows the Content of the mail.

To complete the information needed by an interactive user interface, the next section presents the Orchestration model that incorporates the dynamic behaviour of the GUI widgets that has been defined by the presentation model

5. The Orchestration model

The Orchestration model is a profile of the UML state diagram which captures interaction patterns from presentation widgets as well as the navigation between screenshots of a RIA. The orchestration model allows us to indicate how different widgets receive the events from the user or from other widgets and how these widgets react by sending other events to one or more widgets or by invoking a service offered by the server side. However, not all widgets are liable to be tighted together. Widgets can play different roles in the presentation model. Some widgets just render some static content (e.g. Image, DockPanel), others can realize navigation (e.g. TreeItem, Button). This model has focused on GWT widgets that support a functional unit of interaction (e.g. displaying mail) liable to be orchestrated with other widgets (when its Coordinable attribute has true value). An *Orchestral Widget* provides a unit of interaction with the user to achieve a meaningful task (e.g. sending a mail). Orchestral widgets are the subject matter of the orchestration model.

Figure 6 shows the orchestration model for the GWT Mail case study. We can see that the Mail application has three different screenshots: *Login*, *MailReader* and *MailSender*. In this case, the designer has opt to keep these three Screenshots in only one page. In other cases, the designer could opt to split it into different pages. This model also represents the navigation between these screenshot using transition relationships which are activated by widgets events.

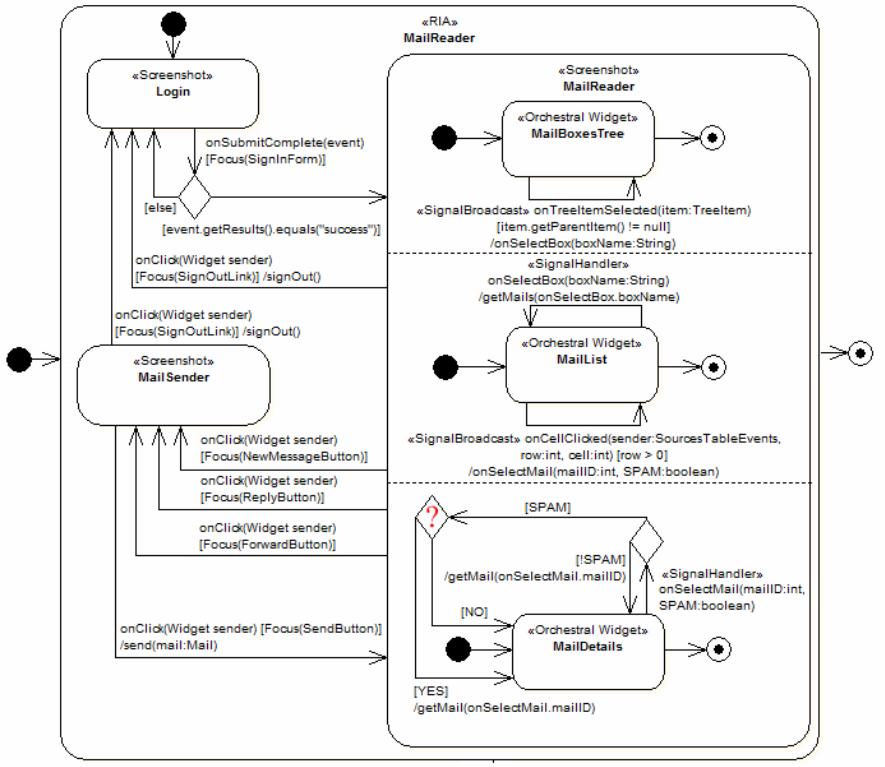

Figure 6. Orchestration model of the GWT Mail application

The orchestration model also defines how widgets interact between each other. For example, *MailReader* screenShot contains three Orchestral Widgets (*MailBoxesTree*, *MailList* and *MailDetails*) that send and receive events between each other. For lack of space, we can only explain the interaction that occurs between *MailBoxesTree* and *MailList* widgets. When a user clicks on a *TreeItem*, it fires a user interface event called *onTreeItemSelected*. This event is captured and only if the clicked TreeItem corresponds to a mail folder (i.e. if the TreeItem is not the root of the Tree, item.getParentItem()!=null), an *onSelectBox* signal is fired. *OnSelectBox* signal has an attribute called boxName containing the name of the selected mail folder. At this point, the *MailList* widget captures this *onSelectBox* signal and calls the *getMails* business logic operation sending, as a parameter, the name of the selected folder. *GetMails* method then returns the list of mails contained in a folder, which is passed on as a parameter. Finally, the result of *getMails* invocation is processed by a callback method that gathers the mail list and updates it in the *MailList* widget.

6. The Transformation model

Following a Model-Driven approach, this RIA development process (see Figure 1) has formalized part of its mapping activities using model transformations. We have selected a metamodel mapping approach to specify these transformations because it allows us to get the information of the different models with just their MOF metamodel. The model transformations are sorted into two different types: (1) model-to-model transformations which obtain the information from one or more models and convert them into one or more models and (2) model-to-text transformations which establish a mapping from the models to the implementation.

In this process, the model-to-model transformations have been defined vertically aiming at merging and lowering the level of abstraction of the models. They reduce the modelling effort while keeping the consistency between models of different levels of abstraction. There are two model-to-model transformations and they are defined by means of the standard transformation language QVT [14]:

Nav2Pres: This transformation defines the different screenshots that make up the presentation of a RIA application. The navigation model is queried in order to identify the links that require navigating to a new page (that is, their isSamePage attribute with true

value). Then, the transformation will create the screenshot elements in the presentation Model.

```
key ScreenShot {name};
top relation NavigationModel2PresentationModel {
        nc1, nc2: String;
        checkonly domain nm: NavigationalModel {
                entryNode = n1:NavigationalNode {
                        name = nc1,
                        source = na:navigationalAssociation {}
                }
        };
        enforceable domain pm: PresentationModel {
                p:ScreenShot { name = nc1, isEntry = true}
        };
        where{
           if (na.samePage = true) NavNodeInSameScreenshot (na, p);
           else NavNodeInNewScreenshot (na, pm);
        }
  }
```

Figure 7. The NavigationModel2PresentationModel QVT relation

Figure 7 shows an example of rule defined in QVT called NavigationModel2PresentationModel. This is the top relation of the Nav2Pres transformation. In the checkonly domain, the relation checks whether there is at least one instance of the NavigationalModel element which has an entryNode pointing to a NavigationalNode with a name and a source link defined by a NavigationalAssociation. Only if all these conditions are satisfied the transformation rule will create a PresentationModel containing a ScreenShot with the same name that the NavigationalNode and its isEntry attribute with true value.

Nav&Pres2Orch: This transformation merges the information of the navigation and presentation models and generates an orchestration model skeleton, thus reducing the modelling effort. To be specific, the transformation obtains two different elements: (1) the transformation gathers the information for creating a screenshot behavioral element for each Navigational Node and (2) a Transition element for each Navigational Association. Finally, from the Presentation Model are obtained the Orchestral Widget elements establishing default behaviour for each GUI widget which has an isCoordinable attribute with true value.

```
Key transition {name};
relation NavAssociation2Transition {
        nc1: String;
        checkonly domain nl: NavigationalAssociation {
                isSamePage = false,
                source = n1:NavigationalClass { name = nc1},
                target = n2set:NavigationalClass {}
        };
        enforceable domain t: Transition {
                source = s1:State { name = nc1},
                target = s2:PseudoState { name = nc2,
                  kind = 'choice', outgoing = tset:Transition{}}
        };
        when { n2set.size() > 1 }
        where{ NavAssociation2SimpleTransition (n2set, tset)}
  }
```

Figure 8. The NavAssociation2Transition QVT relation

Figure 8 shows an example of rule defined in QVT called NavAssociation2Transition. This relation checks whether there is at least one instance of the NavigationalAssociation element which has the SamePage attribute with false value and has target collection with a size > 1. Only if all these conditions are satisfied will the transformation rule create a transition whose target is a Pseudostate that has the kind attribute with "Choice" value.

Finally, the model-driven development process defines two activities that represent model-to-text transformations, allowing to generate the implementation of a RIA. To specify these model-to-text transformations, we have selected the MOFScript language which follows the RFP Model to Text (M2T) of the OMG [13].

GWT Server Side: this transformation converts the OOH domain and navigation model into the RIA Server side. This process focuses on the GWT platform, and proposes the conversion to the Java platform. Basically, from the domain model we can generate the relational database and the business logic offering the CRUD methods (create, read, update and delete) gathered from the domain model and the presentation logic, that is, the server side of the navigation is obtained by the navigation model. We would like to point out that the work is focused on the client side, and that the mapping from OOH to code, is based on previous works [5]. However, our approach defines the transformation rules in MOFScript instead of in Python.

```
om.OrchestralWidget::mapSignalSenderTransitions2GWTEvent){
 var transitions = self.getTransitions()
 transitions->forEach (trans){
    var transParams = trans.convertParameters2Java()
    <% public void %> trans.getName() <%(%> transParams<%){%>
      if (trans.hasGuard() == true)
        <% if (trans.transformOCLGuardToJava()) {%>
      var signal = trans.getSignal()
      var signalParams = signal.convertParameters2Java()
      signal.getName() <%signal new %> signal.getName()
            <%(%> signalParams <%);%>
      <%MessageBroker msg = MessageBroker.getInstance();%>
      <%Event event = new Event (%> signal.getNameSpace
           <%,%> signal.getName()<%);%>
      <%msg.publishEvent (event);%>
      if (trans.hasGuard() == true) <%}%>
   <%}%>
 }}
```

Figure 9. The mapSignalSenderTransition2-GWTEvent MOFScript rule

GWT Client Side: it establishes a transformation from the presentation and orchestration model to the GWT code in order to define a RIA. So that from the presentation model we will obtain the widget composition that makes up the user interface of RIA, and from the orchestration model we will gather the events between these widgets and the invocations to the server. Figure 9 shows a MOFScript rule example of this transformation which generates the events associated with a GWT widget from a Transition element of the orchestration model.

7. Related Work

This section compares our work with related research in the area of Web Engineering where MDE has been applied to the development of RIAs. To do that, we have defined a comparative table (see Table 1) that compares two approaches (OOHDM [15] and RUX+WebML [9]) with the present work and how they apply or not to a set of characteristics belonging to the model-driven Web engineering and RIA disciplines.

Characteristic	OOHDM	RUX+ WebML	OOH4RIA
Domain	Conceptual Model	Data Model (WebML)	Conceptual Model
Navigation	Navigation Model	Hypertext Model (WebML)	Navigation Access Diagram
Structural RIA Presentation	Interface Model	Abstract, Concrete	Presentation Model
Model		(Spatial Presentation) and Final Interface	
Behaviour RIA Presentation	ADV-Chart	Concrete interface (Temporal and Interaction Presentation)	Orchestration Model
Metamodelling	None	Partial (WebML)	Complete
Transformation Model-to-model	None	Code generator	QVT
Transformation Model-to-text	None	code generator	MOFScript
OMG Standards	None	UML	UML, MOF, QVT, OCL
Tool	None	WebRatio & RUX Tool	Eclipse GMF Tool (In development)

Table 1. Comparative table of RIA approaches

According to the order established in table 1, we will start with models that represent the different views of a RIA. As we can see, every approach defines or reuses the Domain and Navigation models coming from traditional Web applications in order to obtain the data and business logic. On the one hand, OOHDM and OOH4RIA have been extended their previous models reusing and introducing new models for RIA. On the other hand, RUX gathers the WebML models as an initialization mechanism to start its RIA methodology.

From here, we compare the models proposed for gathering the structural and behavioural aspects of RIA. The structural RIA models are represented in an Abstract Interface stage by OOHDM and RUX, whereas a concrete interface RIA is tackled by RUX and OOH4RIA.

The RIA behavioural presentation is also dealt with by these approaches but in a different manner. RUX proposes the temporal and interaction presentation models which allow to capture the behaviours that require temporal synchronization and user interactions. OOHDM, on the other hand, defines a ADV-Charts indicating how ADVs are affected when the user interacts. The OOH4RIA Orchestration model is similar to the OOHDM ADV-Charts both allowing to represent GUI widgets interactions. However, the orchestration model introduces a novelty consisting in defining the navigation established between different screenshots and pages allowing us to obtain a multi-page RIA.

At this point, the comparative table shows how the three approaches deal with model-driven issues. As

previously mentioned, metamodelling is used by the OOH4RIA for formalizing RIA models. However, OOHDM and RUX do not define any known metamodel. Another model-driven issue is the vertical mappings aimed at obtaining the RIA implementation. While OOH4RIA proposes a set of model-to-model and model-to-text transformations in order to establish a traceable process. RUX uses code generators to obtain the final implementation.

Another important aspect to take into consideration is the usage of OMG standards, thus accelerating the development of a tool and obtaining better interoperability with other approaches. OOHDM does not follow any standard and RUX only uses UML for WebML models, whereas we use UML, MOF, QVT and OCL languages.

The final row of the comparative table tackles the tool support of different approaches. Up to now, only RUX+WebRatio provides a complete tool, while the OOH4RIA tool based on the Eclipse Graphical Modelling framework (GMF) is currently in a development process and it will be completed in a few months.

For lack of space, other important approaches with RIA extensions like WebML [1] and USIXML [10] cannot be compared in the previous table. These approaches also introduce modelling concepts necessary to generate code of RIA applications.

8. Conclusions

We have focused our study in the OOH4RIA model-driven development process that introduces models and transformations to obtain a complete RIA for the GWT framework. To do that, we have extended a traditional Web methodology like OOH, using its domain and navigation models to define the data and the business logic in order to recover and store this data. From these models, we establish a set of model-to-model transformations obtaining the skeletons of the presentation and orchestration models and reducing human effort and errors.

The presentation and orchestration models are one of the most important contributions of this work. They represent the structural and the behavioural aspects of the RIA user interface, allowing to define both simple and multi-page applications.

However, the evolution of this approach is not finished at all. We started by defining a process aimed at obtaining a promising framework like GWT. However, we are currently working on the definition of an abstract presentation model to represent generic widgets for generating any RIA framework. On the other hand, we are also working on the introduction of the WebSA architectural models [11] into the RIA process, allowing the designer to establish a layer distribution and its different components and connectors.

9. Acknowledgements

This work is co-supported by the Spanish Ministry of Education, and the European Social Fund under contracts TIN2005-05610 (WAPO) and TIN2007-67078 (ESPIA). Perez enjoys a doctoral grant from the Basque Government under the “Researchers Training Program”.

10. References

[1] Bozzon, A.; Comai,,S.; Fraternali,P; Toffetti Carughi, G. Conceptual Modeling and Code Generation for Rich Internet Applications. ICWE 2006, Menlo Park, California, USA, 2006.

[2] Cachero C., Melia S., Genero M., Poels G., Calero C. Towards improving the navigability of Web applications: a model-driven approach. European Journal of Information Systems, 16, 420-447. ISSN 0960-085X/07, 2007.

[3] Driver M., Valdes R., Phifer G. Rich Internet Applications are the next evolution of the Web. Technical Report, Gartner, 2005.

[4] Gamma E., Helm R., Johnson R., Vlissides J. Design patterns: elements of reusable object-oriented software. Reading Mass: Addison-Wesley, 1995.

[5] Gómez J., Cachero C., Pastor O. Conceptual Modeling of Device-Independent Web Applications. IEEE Multimedia, 8(2), 26–39, 2001.

[6] Google Web Toolkit. http://code.google.com/webtoolkit/overview.html.

[7] GWT Mail Application. http://code.google.com/webtoolkit/examples/mail/.

[8] Kent S. Model Driven Engineering. IFM 2002, LNCS 2335, pp.286 –298, 2002.

[9] Linaje M., Preciado J.C., Sánchez-Figueroa F., "Engineering Rich Internet Application User Interfaces over Legacy Web Models," IEEE Internet Computing, vol.11, no.6, pp.53-59, Nov.-Dec, 2007.

[10] Martinez-Ruiz, F. J., Arteaga, J. M., Vanderdonckt, J., Gonzalez-Calleros, J. M., and Mendoza, R. 2006. A first draft of a Model-driven Method for Designing Graphical User Interfaces of Rich Internet Applications. In Proceedings of the Fourth Latin American Web Congress. LA-WEB. IEEE Computer Society, Washington, DC, 32-38, 2006.

[11] Meliá S., Gomez J. The WebSA Approach: Applying Model Driven Engineering to Web Applications. Journal of Web Engineering ©Rinton Press. Vol. 5, No. 2, pp. 121-149. ISSN: 1540-9589. http://www.rintonpress.com/journals/jwe, 2006.

[12] Mesbah A., and van Deursen A., “Migrating Multi-page Web Applications to Single-page Ajax Interfaces”, Delft

University of Technology SERG, Netherlands, TUDSERG-2006-018, 2007.
[13] OMG. MOF to Text Transformation Language Final Adopted Specification. OMG doc. ptc/06-11-01.
[14] OMG. MOF Query/Views/Transformations Draft Adopted Specification: OMG doc. ptc/05-11-01.
[15] Urbieta, M., Rossi, G., Ginzburg, J., Schwabe, D. Designing the Interface of Rich Internet Applications. In Proceedings of the 2007 Latin American Web Conference, 2007.