Personalizing the Interface in Rich Internet Applications

Irene Garrigós[1], Santiago Meliá [1], Sven Casteleyn [2]

[1] Universidad de Alicante, Campus de San Vicente del Raspeig, Apartado 99 03080
Alicante, Spain
{igarrigos, santi}@dlsi.ua.es
[2] Vrije Universiteit Brussel, Department of Computer Science, WISE, Pleinlaan 2, 1050
Brussel, Belgium
Sven.Casteleyn@vub.ac.be

Abstract. Recently, existing design methodologies targeting traditional Web applications have been extended for Rich Internet Application modeling support. These extended methodologies currently cover the traditionally well-established design concerns, i.e. data and navigation design, and provide additional focus on user interaction and presentation capabilities. However, there is still a lack of design support for more advanced functionality that now is typically offered in state-of-the-art Web applications. One yet unsupported design concern is the personalization of content and presentation to the specific user and his/her context, making use of the extra presentational possibilities offered by RIAs. This article addresses this concern and presents an extension of the RIA design approach OOH4RIA, to include presentation personalization support. We show how to extend the RIA development process to model the required personalization at the correct level of abstraction, and how these specifications can be realized using present RIA technology.

1. Introduction

Due to the growing demand for Web applications offering a rich user experience, traditional Web applications are being replaced by the so-called Rich Internet Applications (RIAs), which provide an interface, interaction and functionality capabilities similar to desktop applications. RIA development has new requirements and concerns come into play [17], complicating the task of a Web engineer. The Web engineering community is well-aware of these difficult challenges, extending the design methodologies that target traditional Web 1.0 applications to also support RIAs (e.g. WebML [3], RUX [13], OOHRIA[9], OOHDM [16]). However, due to their relative youthfulness, these new methodologies do not yet cover all design concerns usually encountered in state-of-the-art Web applications. One yet unsupported aspect is the personalization of content and presentation to the specific user and his/her context, specifically for RIAs. RIA UIs are typically dependent on the context device rendering them and vulnerable to the limitations they impose: limited screen size, more difficult interaction and poorer multimedia support. In this paper, we aim to overcome some of these problems by personalizing the UI depending on the specificities of the device (i.e. the device context). To do so, we adapt the UI in two ways: (1) an interface re-organization to fit the UI layout to the device dimensions, and (2) the transformation of some origin widgets into specific widgets that work more efficiently on the target device.

In this paper, we thus extend the existing RIA design method OOH4RIA [9,12] to support the personalization of the RIA user interface for different devices. Based on a set of models and transformations, OOH4RIA defines a model-driven development process that allows to easily introduce new concerns to RIA development. In this paper, we extend the OOH4RIA process by (1) introducing a user model, a device model and a presentation model marking and a widget mapping, (2) defining transformations able to generate presentations for different devices, reducing the effort to redefine new presentation models for each device and (3) integrating a new role in the engineering process called the personalization designer, in charge of defining the personalization models and artifacts. This extended process allows us to obtain different device-aware versions of the same RIA project. The remainder of this paper is organized as follows. Section 2 presents the extensions done in OOH4RIA to integrate personalization. Section 3 presents the main contribution of the paper: the personalization of the RIA user interface to different contexts. Section 4 shows a running example to illustrate the proposal. Section 5 points out how personalization for RIA's and for traditional Web applications differs, and outlines related approaches. Finally, Section 6 provides conclusions and future research lines.

2. Integrating Personalization in the OOH4RIA Development Process

OOH4RIA [9] is a proposal whose main target is to cover all the phases of the Rich Internet Application (RIA) lifecycle development. It defines a model-driven process that specifies the artifacts to obtain an almost complete RIA for a GWT framework [7]. This paper presents an extension of the OOH4RIA development process with activities and artifacts that allow us to introduce the personalization concern into RIA development. To represent this extended process (see Fig. 1) we use the OMG standard called Software Process Engineering Metamodel (SPEM)[11]. Specifically, we have selected the SPEM Activity Diagram because it allows us to introduce the sequence of activities with their input and output work products as well as separating the responsibilities of different process roles. However, the model-driven discipline defines a new kind of automatic activities, artifacts and roles that are not represented by the standard SPEM notation. For this reason, we have extended the SPEM profile introducing a *ProcessRole* stereotype able to represent transformation engines 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.

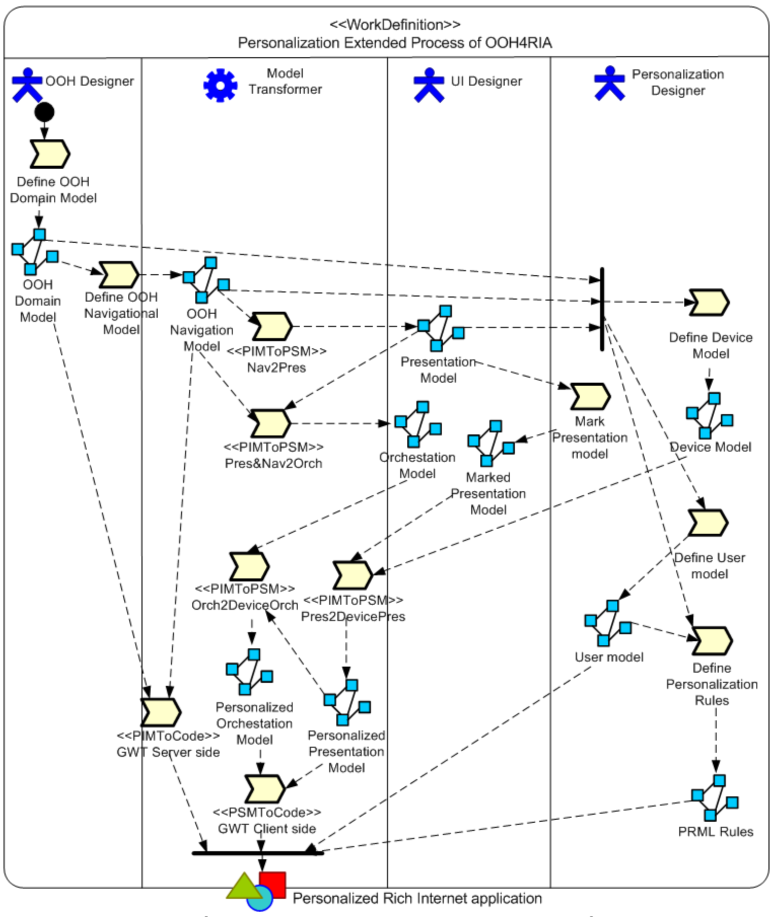
The OOH4RIA process starts when the *OOH designer* defines the OOH domain model in order to represent the domain entities and the relationships between them. This model is the starting point of the three main subprocesses in which this process is split: (1) the definition of the RIA server side where the *GWT Server side* transformation generates the business logic and persistence from the domain and navigational entities, (2) the RIA user interface that begins with the *Define OOH Navigation Model* activity where *OOH designer* represents the navigation through the domain concepts and establishes the visualization constraints using the navigation model. The process continues transforming the navigation model into the presentation model by means of the PIM2PSM transformation called *Nav2Pres* which establishes the different screenshots, which represent spatial distributions of the widgets rendered in a given moment, of the presentation model. After obtaining the container screenshots of the presentation model, the *User Interface designer* completes by placing the widgets, defining the style and establishing the spatial configuration by means of Panels.

The personalization extension introduced by this work presents two initial activities, which can be performed order-independent. The first is the activity *Define Device Model* carried out by the *Personalization Designer* where the domain, navigation and presentation models are the input work product of this activity. In the device model, for each targeted device, a set of widget mappings is specified, which specify the transformation of a particular widget type to another widget type, more suitable for the targeted device type. These mappings can be selected (i.e. re-used) from a predefined set of widget mappings (stored in the so-called Widget Mappings Repository), or newly defined. Depending on the user browsing device, a set of transformation rules defined inside the *Press2DevicePres* transformation (explained in the next paragraphs) will be triggered performing the corresponding widget mappings. The second activity is *Define User Model*, during which the personalization designer specifies the user model. This model represents the dynamic data structures where the information about the user and his/her context is stored which is used to base the personalization on (e.g., device type).

The next step in the personalization process consists of marking the presentation model. The goal of this activity is to mark which elements will be subject to spatial rearrangement, and to provide the necessary details for the subsequent transformation. This process (i.e. marking) has to be repeated for each target device. The markings are defined using the marking technique defined by the MDA guide [10]. This activity produces different sets of markings defined over the presentation model (i.e. marked presentation model), that together with the user model and the widget mappings specified in the device model, are the inputs of the *Pres2DevicePres* transformation.

The personalization process is concluded by the execution of a (fixed) set of transformation rules, part of the OOH4RIA personalization approach. In this work, we focus only on rules referring to presentation issues; content personalization rules are outside the scope of this paper. These transformation rules produce a device-specific presentation model. Concretely, the rules transform the user interface elements that were previously defined in the user interface models in two steps. First, the location of the user interface elements in the target application is transformed, according to the markings made by the personalization designer. Second, widgets are transformed, according to the mappings specified by the personalization designer.

Since the RIA possesses a rich interactive user interface similar to desktop applications, the static features of widgets must be completed 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, where after the designer completes the orchestration model introducing the events, operations and triggers of different states. The orchestration model and the personalized presentation model are the input of the *Orch2DeviceOrch* transformation, which generates a new orchestration model personalized to a specific device, corresponding to re-organized layout and widgets.



<<WorkDefinition>>
Personalization Extended Process of OOH4RIA
OOH Designer
Model Transformer
UI Designer
Personalization Designer
Define OOH Domain Model
OOH Domain Model
Define OOH Navigational Model
OOH Navigation Model
<<PIMToPSM>> Nav2Pres
Presentation Model
Define Device Model
Device Model
<<PIMToPSM>> Pres&Nav2Orch
Orchestation Model
Marked Presentation Model
Mark Presentation model
Define User model
User model
Define Personalization Rules
<<PIMToPSM>> Orch2DeviceOrch
<<PIMToPSM>> Pres2DevicePres
Personalized Orchestation Model
Personalized Presentation Model
<<PIMToCode>> GWT Server side
<<PSMToCode>> GWT Client side
PRML Rules
Personalized Rich Internet application

Fig. 1. Simplified Personalization Extended Process of OOH4RIA

Afterwards, the user model together with the domain, navigation and presentation models are the input that permits to realize the activity *Define Personalization Rules*. This activity defines the personalization rules that establish the different personalization strategies that will personalize the website to the user preferences, goals and context. To define these rules we use the PRML language [6] which was defined in the context of OOH to extend it with personalization support. These rules can be modified at runtime to modify the personalization strategies (this is out of the scope of the present work).

The last step consists of defining the model-to-text transformations that will grant us the personalized 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. These model-to-text transformations are written in the MOFScript language which follows the OMG ModelToText RFP for the representation of model-to-text transformations.

3. Device Context Adaptation of the Presentation Model

In this work, we are focused on the device context personalization of the presentation layer of a RIA, and for this purpose, we must reorganize the layout widgets in the user interface depending on the screen size, and some widgets may need to be transformed into others that better fit the targeted device screen dimensions.

In OOH4RIA, the RIA presentation elements and their layout is represented by the presentation model. As explained, the OOH4RIA presentation model is based on the GWT framework, which is composed of widgets and panels (i.e. layout widgets) where the widgets are placed. For personalization purposes, the designer has to specify how these panels and widgets are transformed and/or reorganized for the target application (i.e. specific device). For instance, one screenshot element (which represent spatial distributions of the widgets rendered in a given moment) specified in the original presentation model may be split into different screenshots in a mobile screen device. As already explained, we allow the personalization designer to add device dependency support using a mark-and-transform approach. By providing pre-defined mappings, and supporting the overall personalization transformation process in the OOH4RIA development process, we significantly reduce the effort for the designer.

The OOH4RIA device context adaptation is made up of following steps:

- *Defining the User Model*, containing user and context variables (e.g., device type), to store runtime information on the user and his/her context. Personalization will be based on the runtime information stored in the user model (e.g., the device type).
- *Defining the Device Model*, by selecting and instantiating existing widget mappings from the Mapping Repository or defining custom mappings. A widget mapping specifies how to transform one widget type (e.g., a Tree) to another widget type (e.g., a MenuBar). The device model consists of different sets of mappings, each targeting a particular device type.
- *Marking the Presentation Model*, in order to determining the spatial arrangement of the panels in the target presentation. As for the widget mappings, different sets markings are specified, each containing markings targeting one particular device type.
- *Perform personalization transformations,* by executing a pre-defined set of transformation rules. The personalization transformation takes as input the presentation model, the user model, the device model and the markings, to generate the personalized presentation model. This transformation is fully

automated, and can be performed at runtime (depending on the device type of the user), or at design time, pre-transforming *n* personalized presentation models, one for each targeted device. In our implementation, we elected the latter approach, to avoid runtime performance overhead.

We now discuss the different steps in more detail.

a) *Defining the User Model*

In the user model, information regarding the user characteristics, interest, preferences or context is stored. In our case, as we focus on device dependency, we store information regarding the device context of the user in order to select, at runtime, which presentation model variant is to be used in the Web application.

b) Marking the Presentation Model

In the following step, the designer marks the presentation model in order to indicate how the elements will be reorganized. For each targeted device type, a new marked presentation model is defined. These sets of markings together with the device model data will drive the transformation rules (explained next), which modify the spatial arrangement of the elements, and transform widgets from one type to another.

To allow the designer marking the elements, the metamodel of the presentation model is extended: each of the panels has a new attribute called *Location* which indicates whether the panel will be placed in a new screenshot or it will be shown in an existing one (in Section 5 we can see an example on marking a presentation model). The location attribute can have different values:

- **inherits***:* this is the default value for all the panels. The panels are nested, all the nested panels will be placed in the same screenshot as their upper panel unless the designer specifies a different value.
- **new***:* in this case the designer specifies that the panel will be placed in a separate screenshot.
- **none***:* this value is assigned when the designer wants to exclude the panel, so it will not be visible (and all what is contained in it) from the target application.
- **all**: the designer assigns this value when he wants to include this panel in all the screens of the target application.
- **containerID**: this value denotes another concrete panel (designated by its containerID) in which the current panel should be nested.

c) *Defining the Device Model*

In order to deal with the personalization at widget level, the personalization designer selects widget mappings from a predefined set of mappings, and instantiates them, complementing them with the necessary details for the specific presentation model (e.g., height and width of an element), If needed, he can define new, custom widget mappings. This results in several sets of mappings, each targeting a specific device type. Each mapping specifies the conversion of a widget (type) to another widget, giving it similar functionality in the target device. In Fig. 3 we can see one set of mappings, targeting a vertical type mobile device for the running example RIA of Section 5.

d) *Performing the Personalization Transformation*

The activity diagram represented in Fig. 3 establishes the general execution workflow of transformation rules that constitute the *Pres2DevicePres* transformation introduced in the OOH4RIA process (see Fig. 1). A fixed set of transformation rules automatically transforms the presentation model to a personalized presentation model. The input of the transformation is the defined set of widget mappings for a specific device and a marked presentation model, which steer the generic transformation process indicating which widgets to transform, and how to relocate panels. As there are *n* sets of widget mappings, and *n* presentation markings, the *Pres2DevicePres* transformation is performed *n* times, each producing a specific personalized presentation model for a specific targeted device type.

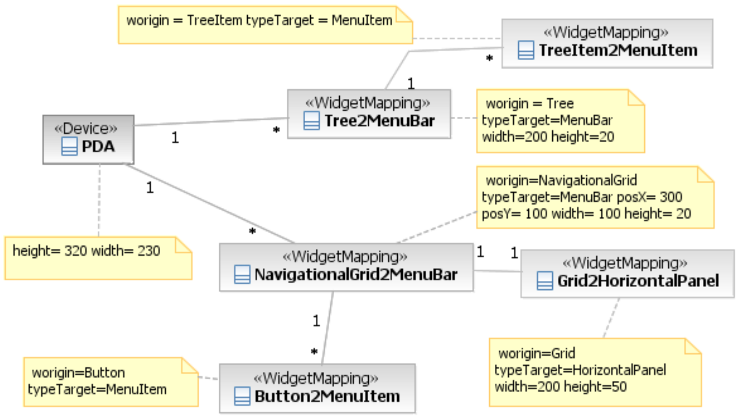
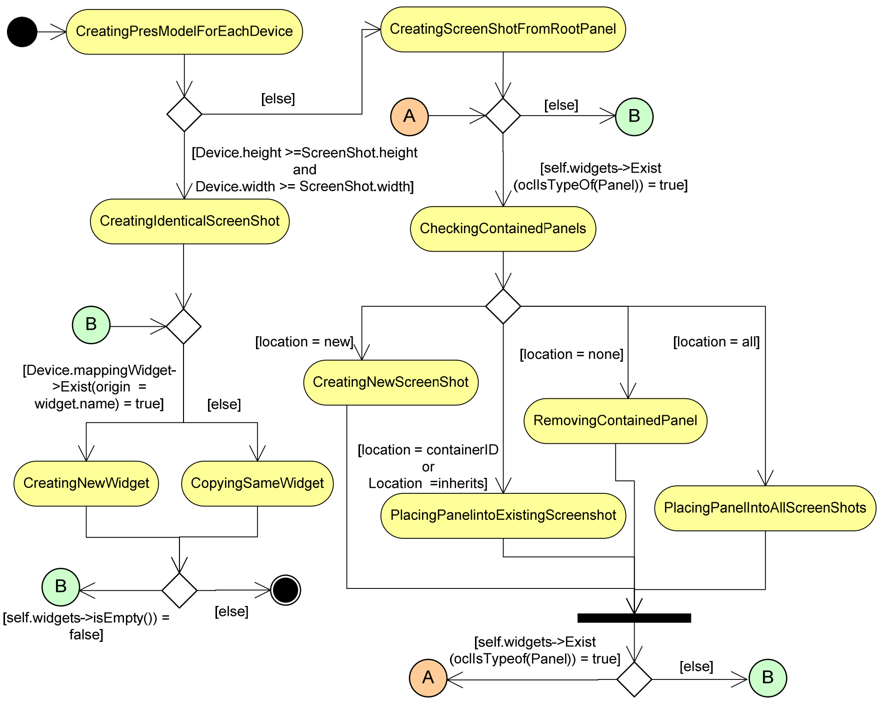

worigin = TreeItem typeTarget = MenuItem
«WidgetMapping» TreeItem2MenuItem
«WidgetMapping» Tree2MenuBar
worigin = Tree typeTarget=MenuBar width=200 height=20
«Device» PDA
worigin=NavigationalGrid typeTarget=MenuBar posX= 300 posY= 100 width= 100 height= 20
height= 320 width= 230
«WidgetMapping» NavigationalGrid2MenuBar
«WidgetMapping» Grid2HorizontalPanel
worigin=Button typeTarget=MenuItem
«WidgetMapping» Button2MenuItem
worigin=Grid typeTarget=HorizontalPanel width=200 height=50
1
*

Fig. 2. The Device Model of GWT Mail application

The execution starts with the root rule called *CreatingPresModelForEachDevice* which creates the presentation model element for each device defined in the device model. When the dimensions (height and width) of the device are larger than the definition, the transformation invokes the *CreatingIdenticalScreenShot* rule, which creates Screenshots identical to the destination model. On the contrary, if the device dimensions are smaller, than the *CreatingScreenShotFromRoolPanel* rule establishes a Screenshot from the container panel with the dimensions adjusted to the device.

Here begins the reorganization of the containers or panels where the transformation checks whether the root panel contains inside panels. If it does, the *CheckingContainedPanels* rule is executed and it decides the destination of the panel according to the value of the location attribute. (1) If location is equals to *new* then the panels requires a new Screenshot, thus executing the *CreatingNewScreenshot* rule. (2) If the location is equal to the *ID* of a pre-existing panel or is equal to inherits then a new Screenshot will be created within it. (3) However, if we want to eliminate the panel (location equal to *none*), we execute *RemovingContainedPanel* rule. (4) Finally, if we want the panel to appear in all the Screenshots (location equal to *All*), the *PlacingPanelScreenshot* rule is executed.

Figure 4 gives specific details of the rule *CreatingNewScreenShot* using the QVT graphical notation. There are two checkeable domains, i.e. two metamodels are checked to see if these domain patterns comply with them.



CreatingPresModelForEachDevice
[else]
CreatingScreenShotFromRootPanel
A
[else]
B
[Device.height >=ScreenShot.height and Device.width >= ScreenShot.width]
[self.widgets->Exist (oclIsTypeOf(Panel)) = true]
CreatingIdenticalScreenShot
CheckingContainedPanels
B
[location = new]
[location = none]
[location = all]
CreatingNewScreenShot
[Device.mappingWidget->Exist(origin = widget.name) = true]
[else]
RemovingContainedPanel
[location = containerID or Location =inherits]
CreatingNewWidget
CopyingSameWidget
PlacingPanelintoExistingScreenshot
PlacingPanelIntoAllScreenShots
B
[self.widgets->isEmpty()) = false]
[else]
[self.widgets->Exist (oclIsTypeof(Panel)) = true]
A
[else]
B

Fig. 3. Rule Map of the ObtainSpecificDevicePres QVT Transformation

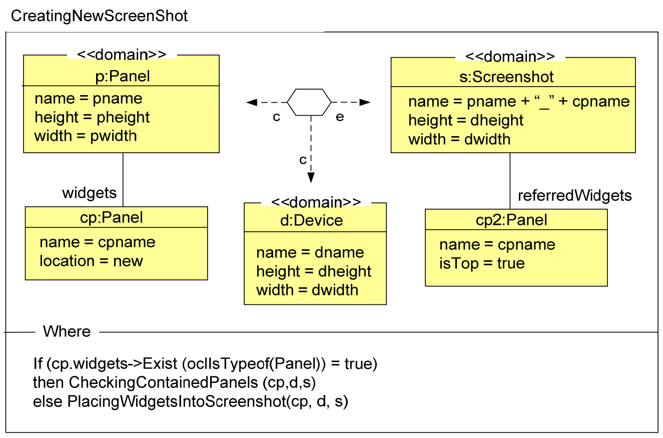
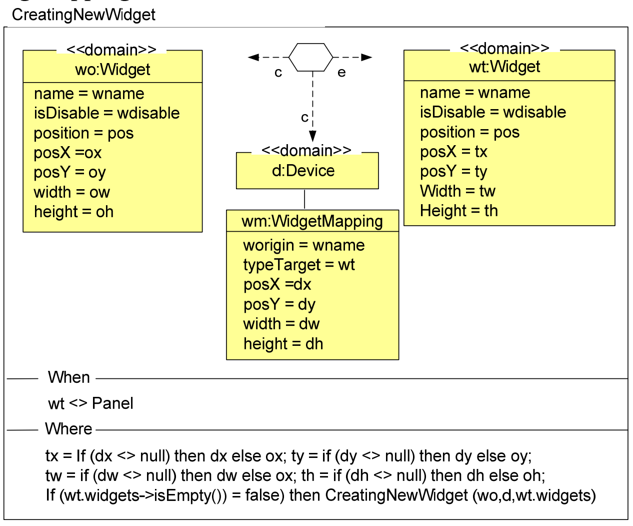


Fig. 4. Example of Pres2Device: CreatingNewScreenShot QVT Transformation Rule

On the one hand, the rule confirms that there is a *p* panel containing a *cp* panel, its location attribute being new. Also, it confirms that there is a Device element in the device model, from which we request name and dimensions. A Screenshot element is now created in the enforceable domain, with the name derived from the panel and its contained-panel (pname + "_"+ cpname). Additionally, the rule creates a root panel called cp2 inside this Screenshot containing the elements of the original Screenshot. Finally, the rule executes the where part which checks whether the panel in its turn contains other panels. If this is the case, the rule recursively invokes the *CheckingContainedPanel*s rules; otherwise, the workflow goes to point B starting the simple Widgets transformation side.

Point B in the *Pres2DevicePres* transformation is where the widget transformations start. Here, the transformation checks if there is a WidgetMapping into the device model for the current Widget. If it does not, the original Widget is copied into the target presentation model by the *CopyingSameWidget* rule. If it does, the *CreatingNewWidget* rule is executed, which transforms the widget according to the corresponding mapping from device model.



```
wt <> Panel
```

```
tx = If (dx <> null) then dx else ox; ty = if (dy <> null) then dy else oy;
tw = if (dw <> null) then dw else ox; th = if (dh <> null) then dh else oh;
If (wt.widgets->isEmpty()) = false) then CreatingNewWidget (wo,d,wt.widgets)
```

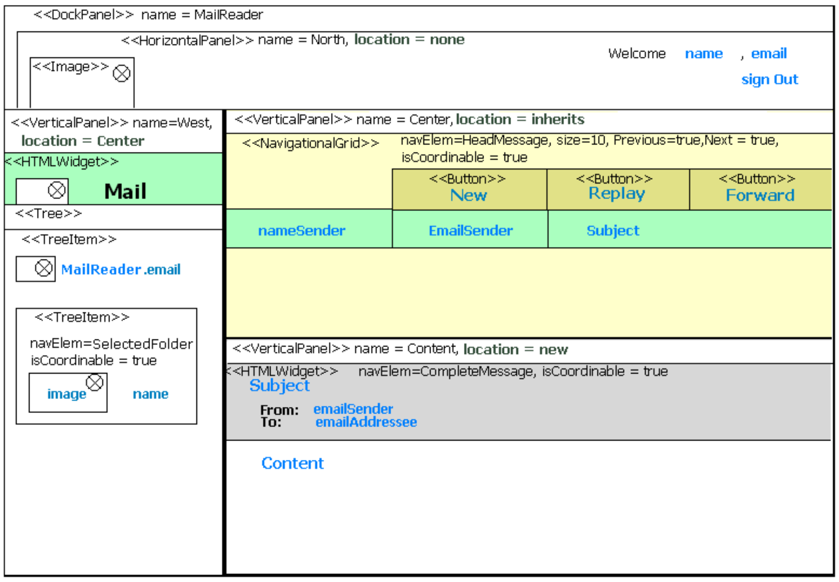
Fig. 5. CreatingNewWidget QVT Transformation Rule

Figure 5 presents the *CreatingNewWidget* rule, converting one widget into another one by gathering the information from the WidgetMapping defined in the device model. Firstly, this rule checks that the source Widget is not a panel in the *When* sentence. From here, the rule creates a new widget that maintains the same name, position and isDisable properties. However, the rule introduces the personalization information from the device model (see Fig. 2), where the WidgetMapping defines a new Widget by means of the typeTarget attribute, and establishes the new location of the widget with the posX and posy, and the new dimension with height and width attributes. Finally, the rule checks if the Widget contains other nested Widgets in the Where clause of the QVT rule, in this case, this rule is invoked recursively in order to transform the contained widgets.

In the next section we present the different artifacts generated during the process applying them to a clear and simple case study: the GWT Mail application [12]. In essence, this case study demonstrates how to construct a relatively complex user interface, similar to many common email applications, and how to easily adapt this user interface to a smaller device such as PDA.

4. Running example

Figure 6 shows the presentation model for the GWT Mail case study. This example was also used in [9] where a complete description of all the design models is presented. In this paper we only focus on the presentation model and we are going to adapt this model so it is suitable for a mobile device. To model the needed device personalization the designer has to follow the steps explained in Section 3:



<<DockPanel>> name = MailReader
<<HorizontalPanel>> name = North, location = none
<<Image>>
Welcome name , email
sign Out
<<VerticalPanel>> name=West, location = Center
<<HTMLWidget>>
Mail
<<Tree>>
<<TreeItem>>
MailReader.email
<<TreeItem>>
navElem=SelectedFolder isCoordinable = true
image
name
<<VerticalPanel>> name = Center, location = inherits
<<NavigationalGrid>>
navElem=HeadMessage, size=10, Previous=true,Next = true, isCoordinable = true
<<Button>> New
<<Button>> Replay
<<Button>> Forward
nameSender
EmailSender
Subject
<<VerticalPanel>> name = Content, location = new
<<HTMLWidget>> navElem=CompleteMessage, isCoordinable = true
Subject
From: emailSender
To: emailAddressee
Content

Fig. 6. Marked Presentation Model of the GWT Mail application

a) Defining the User Model

The user model is defined storing information regarding the device context of the user in order to select, at runtime, which presentation model variant is to be used in the Web application. In our case, the user model contains a variable “DeviceType”, storing the device type of the user.

b) Marking the Presentation Model

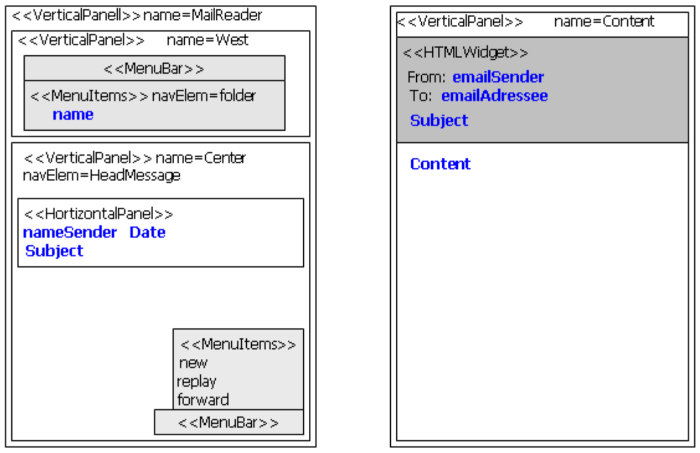
Following the OOH4RIA development process, the designer marks, for the targeted device, the different panels depending on where he wants to locate them in the target application. In this case, the designer decides the *north* panel will be eliminated in the target website; the *center* panel will inherit[1] the location of its upper panel (in this case the rootpanel called *MailReader*). The *content* panel will be located in a new screenshot. Finally, the *west* panel is relocated in the same screenshot as the *center* panel (so the location value is its containerID, *e.g center*). The marked presentation model for the running example is shown in Fig.6.

[1] As explained before *inherits* is the default value so it is not needed to explicitly specify it, but we show it here for clarity purposes.

c) Defining the Device Model

After marking the presentation model, the designer specifies the transformations for the desired widgets. In our example, he instantiates the existing TreeToMenuBar mapping from the Mapping Repository and specifies the target position of the widgets (in the target panels) and their size. Furthermore, he defines a custom mapping to transform a NavigationalGrid to a MenuBar.

The device model for this case study is shown in Fig 2. In this case two main *WidgetMapping* classes are specified, namely *Tree* (originated from the Mapping Repository) and *NavigationalGrid*. These classes represent the *tree* widget located in the vertical panel called *west* of the source presentation model and the *navigationalGrid* widget located in the vertical panel called *Center* of the same model respectively. In the case of the *tree* widget the designer specifies that it is transformed into a *MenuBar* widget. In the same way, the widgets contained in the *tree* (i.e. *TreeItems*) are mapped into *MenuItems*. In the case of the *NavigationalGrid* widget, the designer specifies its target widget as a *MenuBar* widget. The *NavigationalGrid* is a custom widget (the reader can consult the metamodel of the presentation model in [8]), and in this case it contains other widgets to be mapped: *buttons* which are mapped as *MenuItems*, and a *Grid* that is mapped into a *Horizontal* panel.



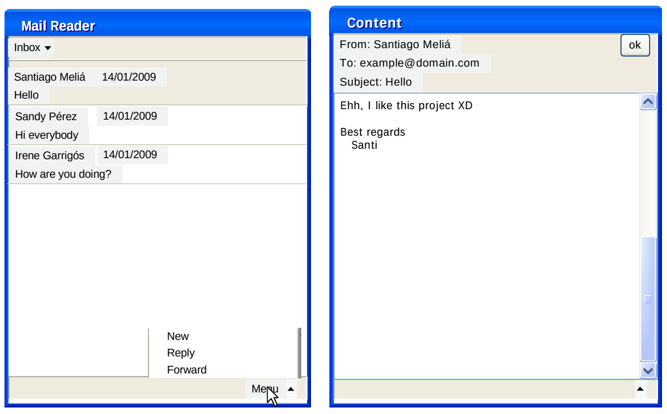
<<VerticalPanell>> name=MailReader
<<VerticalPanel>> name=West
<<MenuBar>>
<<MenuItems>> navElem=folder
name
<<VerticalPanel>> name=Center navElem=HeadMessage
<<HortizontalPanel>>
nameSender Date
Subject
<<MenuItems>>
new
replay
forward
<<MenuBar>>
<<VerticalPanel>> name=Content
<<HTMLWidget>>
From: emailSender
To: emailAdressee
Subject
Content

Fig. 7. Screenshots of the Presentation Model of the GWT Mail for PDA

d) Performing the Personalization Transformation

Given the marks done in the presentation model and having the device and user models specified, the set of transformation rules specified in Fig. 3 is executed. First, as the screen of the target device is smaller than the one of the source device, a new screenshot is created from the root panel called *MailReader (CreatingScreenShotFromRootPanel rule).* The next step checks the panels of the source presentation model based on the marks done by the designer. For the upper panel (i.e. *north*) the location is *none*, so the *RemovingContainedPanel* rule is executed. This panel will not be present in the target application. As such, the marks of all panels are inspected, applying the appropriate transformation rules. For example, the panel called *content* has location=*new* which means that the rule

CreatingNewScreenShot is performed, creating a new screenshot where the panel *content* is placed.



Santiago Meliá 14/01/2009
Hello
Sandy Pérez 14/01/2009
Hi everybody
Irene Garrigós 14/01/2009
How are you doing?

From: Santiago Meliá
To: example@domain.com
Subject: Hello

Ehh, I like this project XD

Best regards
Santi

Fig. 8 ScreenShots of the implementation of the GWT Mail for PDA

The vertical panel called *center* inherits its location from the upper panel (*MailReader)* so it is placed in the same screenshot (already created). Finally the vertical panel called *west* is also located in the same screenshot because its location is the same as the *center* panel (*i.e. containerID=center*).

After the transformation of panels is performed, widgets are checked. In this case the indicated widgets (*Tree* and *NavigationalGrid*) are transformed in the device model performing the *CreatingNewWidget* rule. In Fig. 7 we see the generated presentation model and in Fig.8 two screenshots of the generated application.

5. Related Work

Personalization has been intensively studied in traditional Web application methods. Typically, content, navigation and presentation are personalized to tailor to the specific user based on his/her preferences, characteristics, context and browsing behavior. Traditional Web applications limit the possibilities to track the user browsing to the requests performed to the server. RIAs provide new client-side capacities, new presentation features and different communication flows between the server and client side. These differences with respect to traditional Web applications must be taken into account in RIAs design, as well as in the specification of personalization strategies. Moreover due to the richer set of events that can be contemplated (e.g. drag and drop, scroll, mouse over, etc…) interaction with the user gets richer too and as a consequence, new and more accurate personalization possibilities arise.

In this paper, we focus on the presentation layer, where a RIA website gets more distinctive from a traditional website. In the context of traditional Web applications we can find several approaches treating the personalization of the interface [1, 4, 5, 8]. We highlight two approaches: [1] and [5] in which the layout is personalized depending on the user access device. As explained, presentation in traditional websites is very limited. RIA applications provide richer and more interactive user interfaces, similar to desktop applications. They offer multimedia native support and support animations. As a consequence, from a personalization point of view, the layout and look-and-feel of the application can be personalized but also system reaction to user interaction has to be specified accordingly. Recently, existing Web design methodologies were extended to also support RIAs. The most relevant ones are (1) OOHDM[16] which provides the use of ADVcharts to model widget interaction[14]. (2) WebML which extends its conceptual modeling primitives for RIA's [3] and provides support for distributed event-driven RIA's and specific interaction patterns typically occurring in RIA's[2]. (3) RUX [13], a method independent presentation framework for RIA's, allowing it to tackle presentational specificities of RIA's. RUX has been applied to WebML and UWE, lending its presentational capabilities to these approaches and (4) OOH4RIA which we will elaborate and use as a framework in this article.

To the best knowledge of the authors, there is only one approach [15] that provides personalization support specifically targeting Rich Internet Applications. This approach is not in the context of Web engineering and performs on-the-fly adaptation over AJAX pages. The authors combine ontologies to annotate RIAs and adaptation rules which are derived from semantic Web usage mining techniques. This approach however, does not contemplate the personalization of the presentation features, which is exactly the focus of this paper. We thus present a personalization approach founded in a Web application method, and specifically focus on the RIA-specific elements of the presentation layer.

6. Conclusions and future work

In this paper, we presented an approach that allows achieving device-dependence in Rich Internet Applications, by extending the existing Web design method OOH4RIA with personalization support. We herein focused on the enhanced presentational capabilities of RIA's. We positioned the personalization design activity in the overall RIA design process, and explained it in detail. Our approach consists of two main steps. During the first step, the personalization designer marks the presentation model for spatial rearrangement of widgets in the targeted device. This marking needs to be done for each targeted device. The second step consists of the definition of the device model, by specifying widget mappings: one set of mappings for each device, specifying which and how the different widgets should be mapped onto other widgets for a targeted device. The designer does so either by selecting existing widget mappings, and instantiate them for particular use (i.e., specifying concrete values for generic parameters, such as height or width), or creating custom ones. Based on the markings and the mappings, specific (device-dependent) presentation models are automatically derived. This is done by a set of transformations specified as part of the OOH4RIA development process.

Our approach was illustrated using a case study, consisting of GWT mail application, which we personalized for (small screen) PDA devices.

Currently, we are integrating the personalization transformation in the OOH4RIA tool, which is based on the Eclipse Graphical Modelling framework (GMF) and supports the overall development process. Furthermore, we are currently working on defining the transformation rules that should be performed over the orchestration model to complement the work described here.

References

1. Carughi, G. T., Comai, S., Bozzon, A. and Fraternali, P.: Modeling Distributed Events in Data-Intensive Rich Internet Applications. In 8th International Conference on Web Information Systems Engineering, 2007.
2. Comai, S. and Carughi, G. T. A Behavioral Model for Rich Internet Applications. In 7th International Conference on Web Engineering, 2007.
3. Dolog, P. and Stage, J. Designing Interaction Spaces for Rich Internet Applications with UML. In 7th International Conference on Web Engineering (ICWE), 2007.
4. Fiala, Z., Frasincar, F., Hinz, M., Houben, G.J., Barna, P. and Meissner, K.. Engineering the presentation layer of adaptable web information systems. In Web Engineering 4th International Conference (ICWE), 2004,
5. Garrigós, I. A-OOH: Extending Web Application Design with Dynamic Personalization, Phd thesis, University of Alicante, 2008.
6. Google. Google Web Toolkit (GWT). Online at http://code.google.com/webtoolkit.
7. Houben, G.J., Van der Sluijs, K., Barna, P., Broekstra, J., Casteleyn, S., Fiala, Z., Frasincar, F. Hera, Chapter 10 in Web Engineering: Modelling and Implementing Web Applications, Human-Computer Interaction Series, 2007.
8. Martínez, F. J., Muñoz, J. Vanderdonckt, J. and González, J. M. A First Draft of a Model-Driven Method for Designing Graphical User Interfaces of Rich Internet Applications. In 4th Latin American Web Congress (LA-Web), 2006.
9. Meliá, S., Gómez, J., Pérez, S., Diaz, O. A Model-Driven Development for GWT-Based Rich Internet Applications with OOH4RIA. Eighth International Conference of Web Engineering, Yorktown Heights, USA, 2008.
10. Object Management Group (OMG). MDA Guide: (version 1.0.1). Published at www.omg.org/docs/omg/03-06-01.pdf, June 2003.
11. Object Management Group (OMG). Software Process Engineering Metamodel, version 1.1. Published at http://www.omg.org/docs/formal/05-01-06.pdf, 2005.
12. Pérez, S., Díaz, O., Meliá, S. and Gómez, J. Facing Interaction-Rich RIAs: The Orchestration Model, 8th International Conference of Web Engineering, USA, 2008.
13. Preciado, J.C., Linaje, M., Comai, S. and Sánchez- Figueroa, F. Designing Rich Internet Applications with Web Engineering Methodologies. In 6th International Conference on Web Engineering, 2006.
14. Rossi, G., Urbieta, M., Ginzburg, J., Distante, D., Garrido, A. Refactoring to Rich Internet Applications. A Model Driven Approach. 8th International Conference of Web Engineering, USA, 2008.
15. Schmidt, K., Stojanovic, L., Stojanovic, N. and Thomas, S. : On Enriching Ajax with Semantics: The Web Personalization Use Case. ESWC '07: Proceedings of the 4th European conference on The Semantic Web. Innsbruck, Austria, 2007.
16. Urbieta, M., Rossi, G., Ginzburg, J. and Schwabe, D. Designing the Interface of Rich Internet Applications. In 5th Latin American Web Congress, 2007.
17. Wright, J. M., Dietrich, J.B.: Requirements for Rich Internet Application Design Methodologies. Proceedings of the 9th international conference on Web Information Systems Engineering, (WISE), 2008.