User Interface Design for Responsive Web Applications

Hernán Casalánguida and Juan Eduardo Durán
Facultad de Matemática, Astronomía y Física, Universidad Nacional de Córdoba, Medina Allende s/n, Córdoba, Argentina
hcasalan@hal.famaf.unc.edu.ar, duran@mate.uncor.edu

Keywords: Web Engineering, Rich Internet Applications, User Interface Models, Responsive Frameworks.

Abstract: The design of web applications that adapt to different kinds of devices is now a necessity. The responsive web design (RWD) is an actual approach to this problem. There exists a large quantity of responsive frameworks (RF) for developing RWDs. In particular for the domain of Rich internet applications and for the adaptation of applications to different kinds of devices we have found a few adaptive design approaches that start with abstract user interface (UI) models; however, such approaches did not take into account the use of RFs. The problem of defining a development process from an abstract UI model to a RF is interesting due to some reasons; a good process should consider: an abstract UI model whose elements are abstractions for RF widgets, the use of tools for RFs that generate part of the final UI code, the use of model transformations to map abstract UI elements onto widgets of the RF. In this paper we created an abstract UI model called RIAAD2 that considers abstractions for all the UI elements of a selected set of RFs, and we developed a process for the creation of a final UI using a RF that considers the above requirements.

1 INTRODUCTION

Due to the dramatic increase in the amount of internet accesses from mobile devices and tablets, the design of web applications that adapt to different kinds of devices (e.g. cell phones, tablets, laptops, desktops) is now a necessity.

During the last years RWD (see (Peterson, 2014)) has become an efficient solution to these problems; with this kind of design it is provided for the users of a web site the same content and a similar user experience; this reduces costs and time to market, because a RWD of a web application works across all devices. Now there exists a large quantity of RFs (e.g. Bootstrap, JQuery Mobile, HTML Kickstart, Foundation, Skeleton) which allow to develop RWD for web applications.

For the domain of Rich internet applications (RIA) and for the adaptation of applications to different kinds of devices we have found only a few adaptive design approaches (i.e. in them a server detects the device, and the browser will load the version of the site that is optimized for that device; i.e., only mobile-optimized assets are downloaded) that start with abstract UI models: (Cirilo et al., 2012), (Manca, 2013) and (Ghiani et al., 2014); however, such approaches do not consider RFs.

The problem of defining a development process from an abstract UI model to a RF is interesting due to the following reasons: 1) the wide use of RF in industry, 2) responsive applications do not require an additional architecture at server side, 3) responsive applications are applications for internet (complex architectures at the server side for making adaptations are adequate for intranets), 4) when a modification is needed, a new version of the responsive application is constructed (in the other approaches it will be necessary to generate the code for each device again). Such a process should satisfy the following requirements:

R1: the use of an abstract UI model whose elements are abstractions for widgets of RFs.

R2: the use of tools for RFs that generate part of the code for the RF.

R3: the use of the model transformation approach to map UI elements of the abstract UI model onto widgets of the RF.

We did not find a UI design notation that is an adequate abstraction of the most important RFs; in addition, we did not find a method for the construction of a final UI considering a selected RF and an abstract UI model.

The objectives of this work are: a) to develop an abstract UI notation for RIAs, such that the widgets

of the best RFs (according with some criteria) are refinements of the UI elements of this UI notation, and this UI notation abstracts from implementation details, development technology and target device, and is independent from modality; b) to define a development process contemplating the above requirements, and the code considers at least widgets and layout.

In this paper we selected the most successful and useful RFs (according with some criteria) (Sec. 2); to satisfy R1 we created a new version of the RIA Abstract Design notation (see (Casalánguida and Durán, 2013)) called RIAAD2; all the widgets of the selected RFs are refinements of UI elements of RIAAD2 (Sec. 3); furthermore, we defined a development process satisfying the above requirements (Sec 4); for this purpose, we defined a table that maps RIAAD2 UIEs onto widgets of the selected RFs; next using this table we explained how to transform a RIAAD2 UI model into a more useful model to be used during implementation; finally, we provide some tasks using this model, and a RF's tool to construct the final UI.

For illustrating the RIAAD2 notation and the development process we considered a case study consisting of a system of online file storage.

2 RESPONSIVE FRAMEWORKS

The development of several versions of an application for different devices is usually not a good option, because it is expensive. For this reason, during the last five years, several RF have appeared for the development of responsive web applications. A RF is a framework for RWD.

In this section we evaluate and select RFs to build responsive web applications. For the selection of RFs we considered the following requirements:
a) the RF can be used with all kinds of actual devices (i.e. mobile devices, tablets and desktops);
b) the RF has a rich set of widgets (i.e. widgets for content structures, access structures based on links, access structures not only based on links – i.e. they contain in addition to links, controls for input/items for content- and buttons);
c) the RF includes a responsive grid system for layout. A *responsive grid system* is grid system (i.e. a grid) that appropriately scales up to *N* columns as the device or viewport size increases; it includes predefined classes for easy layout options, as well as powerful mixins for generating more semantic layouts;
d) the RF is popular (i.e. number of search results in web search engines and amount of external libraries defined using the RF).

The following RFs are discarded, because they do not satisfy requirement a): Blueprint- only for desktops-, Flaminwork- only for desktops-, G5 Framework- only for desktops -, Easy Framework - only for desktops -, Elements – only for desktops-, Bluetrip - only for desktops -, ElasticCss – for desktops and mobile devices, but not for tablets. The following frameworks satisfy requirement a): Bootstrap, Mobile Boilerplate, Foundation, jQuery Mobile, HTML Kickstart, Less and Skeleton.

Next, we discarded the following RFs: Less (see http://lessframework.com/), because it is powerful for layout definition, and does not have a rich set of UI widgets (it includes only a responsive grid system and sets of typography presets); Skeleton (see http://getskeleton.com/), because of the scarcity of the widgets (it only includes elements for buttons, forms, grids and typography presets); Mobile Boilerplate (see https://html5boilerplate.com/), because it offers a front-end template for cross-browsing, performance optimization, optimizations for browsers of mobile devices, but it does not include widgets.

The 4 RFs that satisfy the above requirements are: Bootstrap (see (Sossou and Shenov, 2014)), jQuery Mobile (see http://jquerymobile.com/), Foundation (see http://foundation.zurb.com/) and Html KickStart (see http://getkickstart.com/). All of these RF have a rich set of widgets, and include a responsive grid system.

3 AN ABSTRACT UI MODEL

Requirements for an Abstract UI Notation. Given the great amount and variety of RFs, we think it is a good idea to consider an abstraction level above that of RFs; more specifically, it is desirable to have an abstract UI design notation satisfying the following requirements: 1) its elements are abstractions for RF UIEs; 2) it abstracts from layout and style; 3) it is as independent as possible from modality and from implementation technology (this requirement was posed for UsiXML abstract UI models; see (Limbourgh 2004)); 4) it has a rich set of access structures and of content structures (this is to avoid: the necessity to choose between too much RF UIEs for refining an abstract UIE, and the necessity to infer different choices of UIEs of a RF from an abstract UIE); 5) it has a rich set of classes for UIEs used to generalize metaclasses (e.g. content structure

to generalize list, tree and grid); this is to allow future extensions of the abstract UI model.

Some reasons to have an abstract UI model are: a) In (Thevenin, 2001) it is said that the variety of context of use for an application stresses the need for UI abstractions able to factor out details relevant to specific contexts; from these abstractions, it is possible to obtain context specific representations by progressive refinements. b) (Limbourgh 2004) says: "to have abstractions to improve comprehension, reasoning and manipulation of what a UI is". c) To ensure some form of consistency between requirements artifacts and the final UI - see (Puerta, 1997). d) (Cockburn, 2002) says that if software team members spend little time modeling or documenting applications, this becomes a problem when the team is dismantled and other people needs to maintain the software.

Extension of the RIAAD Metamodel. The RIAAD metamodel (see (Casalánguida and Durán, 2013)) satisfies requirements 2) and 3) above, and defines abstractions for several code patterns, basic UIEs, access structures and content structures. Some of the contributions made by RIAAD to the domain of RIAs are: the representation of *editable* UIEs, being them either elementary or content structures; the definition of abstractions for special UI patterns for navigation in RIA like breadcrumb and navigation bar; the representation of UIEs for the edition of multimedia objects – e.g. audio and video- and of documents – e.g. presentations and spread-sheets).

The RIAAD metamodel was not developed taking into account RFs, and for each of the 4 selected RFs: RIAAD has not abstractions for some of the RF's widgets, or RIAAD´s UIEs need to be generalized to be considered as abstractions for some widgets of RFs.

We decided to modify RIAAD to have an appropriate abstract UI model satisfying the requirements above; the method considered for this task is: examine the UIEs of the selected RFs, and for each of them apply the following procedure: if an element of RIAAD is an appropriate abstraction for the RF's UIE, then we are done, else if a RIAAD's UIE could be generalized to obtain an appropriate abstraction for the RF's UIE, then we make this generalization, otherwise we define a new UIE that is an abstraction for the RF´s UIE in the sense of satisfying requirements 2) and 3); finally, we eliminate some UIEs from RIAAD that are specific for code patterns, and are not needed for RFs.

Using this method we developed a new version of RIAAD called RIAAD2; we added to RIAAD 2 the following new UIEs: Icon, Button, Dialog, NavGrid, Grid and Alt; in addition, we generalized from RIAAD: Menu, Breadcumbs, NavigationBar, Content Structure, Block and Grouping Element.

RIAAD2 adds new UIEs not present in the found abstract UI modelling notations for RIAs: Icon, NavGrid, Grid and Dialog. In addition, we changed the name of some RIAAD UIEs.

Related Work. We compare abstract UI presentation notations for RIAs that are independent from modality, implementation technology, abstract from target device, and have metaclasses for UIs to generalize metaclasses (see Table 1); in this set we included the abstract UI notations of the found adaptive design approaches for RIAs.

Table 1: Comparison of UI metamodels.

	R1	**R2**	**R3**	**R4**
MARIA	reg -	reg -	no	Yes
UWE	reg +	reg	no	No
RID	reg -	reg-	no	No
RIAAD2	good	good	yes	No

R1: *Richness of abstract UI elements for content structures*: The AUI metamodel of MARIA (Paternò, et al., 2009) has not content structures. The UWE presentation metamodel for RIAs (Kozuruba, 2010) has presentationGroup, iteratedPresentation-Group and presentationAlternatives, but not trees. The abstract UI model for RIAs of (Cirilo et al., 2012) called RID has not content structures.

R2: *Richness of abstract UI elements for access strucutres*: The abstract UI metamodel of MARIA has not access structures. The UWE presentation metamodel has presentationAlternatives and iteratedPresentationGroup, but not abstractions for breadcumbs and navigation bar. The abstract UI model for RIAs of RID considers TabbedPanel and AccordionPanel, but not abstracttions for navigationBar, NaviGrid, Breadcumbs.

R3: *Designed for consideration of RF widgets:* Only RIAAD2.

R4: *Use of a concrete UI model:* Only MARIA.

The RIA methodologies found do not construct code using a RF.

IFML (http://www.ifml.org/) is a metamodel for expressing the content and the user interaction with the UI in applications. IFML is poor for satisfying R1, R2 and R3. IFML include concepts about context awareness; however, we did not find previous work concerning the adaptation of a RIA application to different devices using IFML.

3.1 RIAAD2 Metamodel

A *BasicUiElement* can be either an *Atomic* element or a *MediaObject*. A *MediaObjects* can be: *Image*, *Video*, *Audio*, *Animation*, *Document*. A MediaObject can be editable or not. An atomic can be: Text, *Numerical*, *Anchor*, *Single Choice*, *Multiple Choice*. *Type of edition* of an Atomic can be: *input* (for information input), *editable* (for information editing) and *no_editable* (for information presentation). Attribute *enabled* says if an Atomic is enabled or not (e.g. if an anchor´s link is enabled for navigation). Attribute *Type* of anchor can be: *classicalLink* (link with another page/document), *bookmarkLink* (a link with a position in the same page) and *commandLink* (its selection initiates an action or task). An anchor contains one or more BasicUiElements different from anchor and with editableType=no-editable. A *Button* represents a button behaving as an anchor when pressed. An *Icon* is a graphic representation of something (e.g. a person or thing) that is symbolic, or is a noted figure.

In Fig. 1 attribute *collapsible* means if the Grouping could be collapsed and expanded. A ContentStructure (CS) may contain selector elements and anchors. A CS can be *editable* (i.e. allowing the edition of some of its contents) or not. The attribute *filterable* means if the CS elements can be filtered w.r.t. a condition provided by the user; this attribute can only be used for lists, grids and trees. An *Alt* represents the presentation of one BasicUiElement/CS from a set of BasicUiElement/ CS. A *Grid* represents the presentation of several rows of one or more types (a Grid may only contains records -via childCS- for describing its rows). A *Tree* represents a tree whose nodes have content. A tree contains internal nodes and leaf nodes; both kinds of nodes contain a text element for the name of the node and optionally one or more BasicUi-Elements. *UIOutputStructure*: - see Fig. 1 – for presenting information to the user. *Notification* means notification of some event, and *Dialog* means a decision request. *NotificationType* is defined as in RIAAD. A *LinksBased* represents a link grouping to access either other UIEs, or performing an action; some specializations of LinksBased are Menu and Breadcumb. A *Menu* contains two or more anchors, and may contain other menus. A *Breadcumb* contains a list of steps; each step has an anchor; Breadcumbs are used to represent navigation paths, whose nodes can be visited by selecting steps.

Fig. 2 shows part of the UI for a *read Mail* function. Some commands are accessible from the *Mail options* menu and from the *Reply* Button. For the headers of the mail we use a *Mail details* anchor of type CommandLink. The body of the mail is an Alt CS with alternative text elements: *Mail* that is the text of the mail, and *Previous mails* that is the text of the mail and the text of the previous mails to which this mail is a reply.

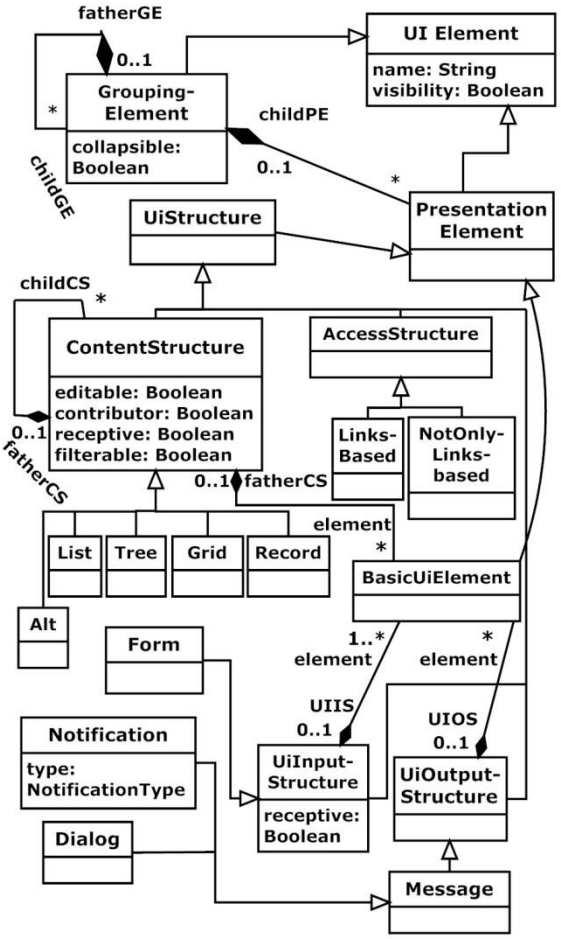

Figure 1: RIAAD2 classification of UI elements and UiStructures.

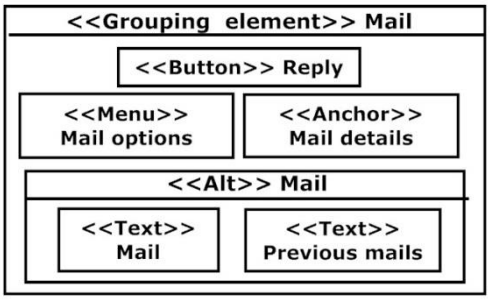

Figure 2: The UI for read Mail function.

A *NotOnlyLinksBased* - see Fig. 3 - is not based only on links. A *NavigationBar* may contain some text elements of NoEditable type. In a *NavAltBlocks*

only one block at a time is visible; at most one block can have zero UiElements; the attribute *enabled* tells if the block is enabled for navigation. A *NavGrid* represents the presentation of several rows of one or more types; a NavGrid contain items for describing its rows; an item contains an anchor with content displayed for an item, optionally a navigationBar for parameters providing and/or functionality access and zero or more BasicUIElements. The attribute *filterable* means if the navGrid items can be filtered w.r.t. a condition provided by the user. A CS/ NavGrid can be a *contributor* (it can provide elements to a CS/UiInputStructure). A CS/UiInput Structure can be *receptive* (it can receive elements from other CS/NavGrid).

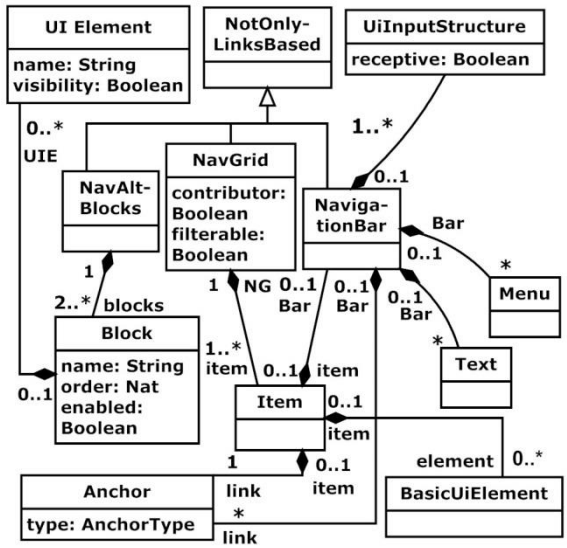

Figure 3: RIAAD2 NotOnlyLinksBased elements.

4 PROCCESS FOR DEVELOPING A FUI USING A RF

Requirements for the development Process. They are: **P1.** It could start with artifacts of other RIA methodologies (e.g., a navigation model – transition from a navigation model to a RIAAD2 model; a presentation model – abstract/refine it to a RIAAD2 model; a requirements model - from it make the transition to a RIAAD2 model).

P2. The process should consider the definition of the abstract UI for the functionality units (e.g. use cases, tasks, commands, services); however, this is not enough, because it is necessary to have a global vision of the UI for functionality access using the metamodel for abstract UI design.

P3. To define onto which widgets of a RF the UIEs of the abstract UI model can be mapped; in case of not exiting such widgets, a mapping of abstract UIEs onto HTML5 elements should be defined.

P4. To develop the code for a RIA application using the RF and a tool for the RF. Such tools are important, because they generate some of the final UI of the RIA application.

Related Work. See Table 2. The criteria are:

Table 2: Comparison of adaptation approaches.

	A1	**A2**	**A3**	**A4**	**A5**
Cirilo et al 2012	no	No	yes	yes	yes
Manca 2013	no	No	No	yes	yes
Ghiani et al 2012	no	no	No	yes	yes
Process for RIAAD2	yes	yes	No	no	no

A1: *Transition from an abstract UI model to a RF*. **A2:** *Use of information for mapping abstract UI elements onto RF widgets*.

A3: *Construction of different UI versions for different groups of devices*. In (Cirilo et al., 2012) from a RID model for a group of devices code is generated for different technologies by applying M2C transformations (they are implemented as templates by using the Java Emitter Templates framework).

A4: *Necesity of additional artifacts to make the transition from an abstract UI model to a final UI model*. (Cirilo et al., 2012) considers the definition of adaptation rules for each UIE of RID to adapt. In (Manca, 2013) the definition of adapters for different purposes is needed (for modality, UI structure and UI attributes) for code generation. (Ghiani et al., 2014) considers the definition of adaptation rules respecting an event/condition /action template using a XML based format.

A5: *Need of an additional server architecture for adaptation*. In (Cirilo et al, 2012) content adapters are used by a server. In (Manca, 2013) an adaptation server is considered. (Ghiani et al., 2014) considers an architecture at the server side for adapting a concrete UI model to a specific device.

Development of AUI Diagrams. To develop an abstract UI two tasks are contemplated: the construction of the UI structures for functionality access, and the construction of the UIs for each functionality unit.

For the UI concerning the structure of the functionality access of the system, access structures like menus, lists and anchors are used a lot.

For the definition of the UI model showing the structural view of the functionality access it can be useful to use previous models of web application methodologies like navigation models (e.g. UWE (Kozuruba, 2010) and others), UI models (e.g. OOWS 2.0 (Valverde Giromé, 2010) and others) or requirements models showing the organization of the functionality of the system (e.g., (Rosado da Cruz, 2010)).

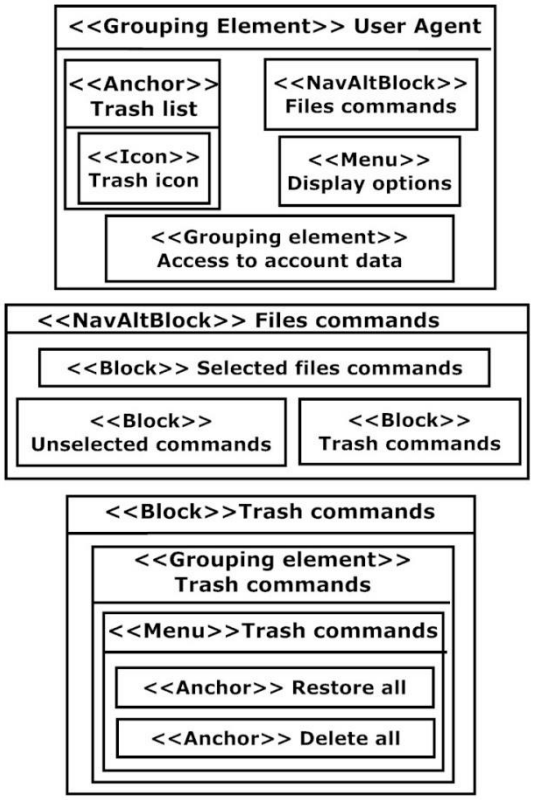

Figure 4: Part of RIAAD2 model for the UI for functionality Access for a system of online file storage.

Fig. 4 shows part of the UI for the structure of the functionality access for the system of online file storage that is shown initially when the user enters into the system. The *User agent* grouping includes structures for the access to functionality: an anchor including an icon to access to the list of items (files and folders) sent to trash; a NavAltBlocks *Files commands* considering different classifications of commands (its blocks are automatically chosen depending on the selected items and its types); a menu *display options* for displaying the list of contents in different ways (i.e. sorting by different criteria, display as icon, and display as list); a grouping for accessing functionality concerning the account of the user. Fig. 4 shows part of the description of the *Files Commands* NavAltBlocks UIE; it consists of four alternative blocks: commands for selected files, commands when no item is selected and commands for the selected elements on the trash. The *Trash commands* Block, includes a menu with *Restore all* and *Delete all* anchors of type commandLink to perform operations for the selected items in the trash.

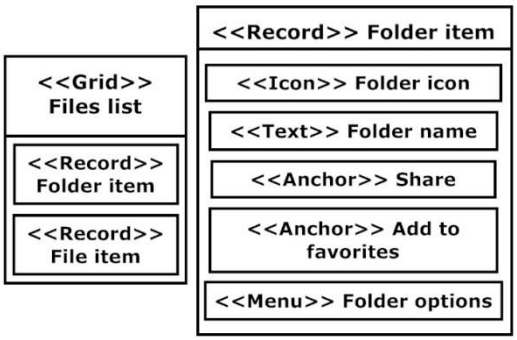

Figure 5: UI for the use case Load and display files list.

It can happen that we have a prior model of a methodology describing some functionality units; this model can be a presentation model, (e.g., UWE (Kozuruba, 2010), MARIA (Paternò et al, 2009)), a model to describe requirements (e.g., task models - e.g., concurrent task trees, see http://www.w3.org /TR/task-models/-, activity diagrams to describe use cases – e.g., (Casalánguida and Durán, 2013)). In (Casalánguida and Durán, 2013), trace relationships from actions in activity diagrams into RIAAD UIEs are considered; therefore, for describing the UI for a UC in RIAAD2 the UIEs of these trace relationships can be reused.

Fig. 5 shows the RIAAD2 model for *Load and display files list* use case that collects a list of files and folders, and presents it with the help of a UI. There is a Grid *Files list* organized into two types of records: for files called *File item* and for folders called *Folder item*. The *Folder item* record includes the icon for folders, the name of the folder and a menu for functionality over folder items. The *File item* record is similar to the *Folder item* record.

Mapping Abstract UIEs onto RF Widgets. Once a RF to be used has been selected, it is necessary to choose the UIEs of the RF to implement the UIE of the abstract UI model. To accomplish this task we constructed a table (it is not shown by space reasons) that for each structural abstract UIE and for each of the four selected RF, provides the final UIEs that can be chosen to implement the (considered) abstract UIE; such final

UIEs are obtained in the following way: if the RF at hand contains Final UIEs that can be used to implement the abstract UIE, then these elements are listed; otherwise a HTML5 element to implement the abstract UIE must be provided by the web designer. For the case that an abstract UIE has more than one corresponding final UIE in the table for the RF, the UI designer has to decide which of these final UIEs is more convenient.

From a RIAAD2 model of the RIA application we propose to automatically generate a new version of the RIAAD2 model having annotations; each UI element of the RIAAD2 model is annotated with a list of widgets of the target RF that are refinements of the abstract UIE; if the list has a unique element then the mapping is direct; else the designer has to select the most adequate widget in the list to be a refinement of the abstract UIE. This RIAAD2 model with annotations is useful, because the designer has not to search the tables to find a mapping; therefore, his work is simplified.

Implementation of the Final UI. Once the decisions concerning the mapping of the abstract UIEs onto final UIEs have been taken, the next step is to implement the final UI of the application. We present some tasks that can be accomplished to implement the final UI of a responsive web application. As a case study to illustrate the use of these tasks we considered Bootstrap and the Pingendo WE free tool (see http://pingendo.com/) that works with Bootstrap.

The tasks we propose to perform are:

a) To create an empty page; e.g. a Bootstrap empty page in Pingendo.

b) Considering the requirements of the client, define the layout of the responsive web application; e.g., use the UIEs of the layout part of the widgets window of Pingendo. In general, for this task it is common to define a Container element of highest level, and consider inside it an arrangement of rows and columns according to the desired layout.

c) Include in each pair (row, column) the UI elements and access structures for functionality access; e.g., for this task Pingendo has the sections *Navigation* and *Buttons*; it is very common that such elements are of kind Button, Button DropDown, NavBar, Breadcumb.

d) For each functionality unit create an empty page, then define the layout, and include the appropriate final UIEs.

e) The abstract UIEs for which there does not exist a corresponding widget for the selected RF must be implemented by using either an external library of widgets or HTML5 elements; e.g., it is necessary to use the part of HTML source code edition of the Pingendo tool.

f) Associate the links for functionality access with the parts of the system that construct the UIs of the corresponding functionality units. This can be a URL for a dynamic page, or a piece JavaScript code.

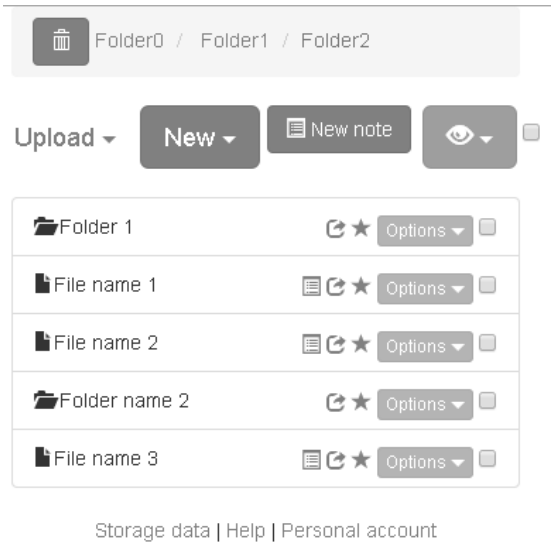

Figure 6: Part of the UI for an online file storage system using Bootstrap with Pingendo.

Fig. 6 shows the UI for the initial screen for an online file storage system presented after the user logged in that was developed using the previous tasks with Bootstrap and Pingendo, and considers the access to the functionality of the system and the UI for two functionality units: *Generate and display files structure* and *Load and display files list.* The UI contains a layout considering one column, and 4 rows: 1) UI design for the *Generate and display files structure* functionality unit using the Breadcumbs widget of Bootstrap. 2) Elements for functionality access for *Files Commands* and *Display Options* RIAAD 2 UIEs; it was built using the following widgets: a Button to create a note, 3 Button Dropdown (one for creating a file or a folder, another to choose how to see the list, and another to upload a file or a folder), and a HTML5 Check Box element for selecting/deselecting all the list items. 3) UI design for the *Load and display files list* functionality unit using a list; each item of this list represents a file or a folder stored by the system, and is implemented with: an icon for the item's type, a text for the item's name, some icons for functionality access, a Button Dropdown widget to access some functionality units, and a CheckBox to select the

item. 4) Access to functionality of *Access to Account Data* RIAAD UIE implemented using anchors.

5 CONCLUSIONS

This work considered responsive web applications from small to big that can be RIAs or not. Our approach considers the definition of UIs for such kind of applications considering UIEs and layout, but not taking into account event processing and style. We start with some abstraction requirements; as a consequence, it is possible to concentrate on requirements and structural aspects of the UI.

Concerning RIAAD2 we have found the following facts:

- 5 RIAAD2's UIEs (Button, Menu with submenu, Dialog, Icon, Grid) are not in RIAAD; this number represent the 50 % of the number of UIEs types used in the case study of this paper; in addition, these 5 UIEs have an occurrence in our case study that represents the 27,8% of the total UIE occurrences.
- From a total of 12 elements that are either not present in RIAAD or modified elements of RIAAD, 8 of them were found when trying to find in RIAAD UIEs that are abstractions of the UIEs of the 4 selected RFs.
- 12 of the 39 UIEs of RIAAD2 are additions to RIAAD or modifications of UIEs of RIAAD, this represents approximately 30%.

Concerning the mapping of RIAAD2 UIEs onto widgets of RFs, there are 3 cases of not Basic-UiElements where a decision about which widget of a RF to use for a UIE of RIAAD is needed: Grid (not table): 2 decisions in average (i.e. considering the 4 selected RFs), menu (without nesting): 5 decisions in average, grouping element: 3.5 decisions in average.

The separation between UI for functionality access and UI for functionality units, the use of the tasks for implementing the final UI and the table for the mapping of RIAAD2 UIEs provide a systematic and disciplined approach to develop final UIs.

Some of the UIEs of RIAAD2 cannot be mapped onto widgets of a given RF (e.g. Tree and Alt in Bootstrap, Foundation and HTML Kickstart); therefore, for these UIEs it is necessary to define source code in HTML5, JavaScript and CSS.

A work for the future is the study of how to migrate from legacy RIA applications to RFs; in particular, we are interested on studying the automatic/semiautomatic transition from abstract UI/concrete UI models for RIAs to RIAAD2.

REFERENCES

Casalánguida, H. and Durán, J. E., 2013. A Method for Integrating Process Description and User Interface Use During Design of RIA Applications. In *ICWE'13, 13th Intl. Conf. on Web Engineering*. Springer Verlag.

Cirilo, C. E.; do Prado, A. F.; Lopes de Souza, W. and Martinez Zaina, L. A., 2012. Building Adaptive Rich Interfaces for Interactive Ubiquitous Applica-tions. "Interactive Multimedia", edited by Ioannis Deliyannis, ISBN 978-953-51-0224-3, Charpter 11.

Clarissa Peterson, C., 2014. Learning Responsive Web Design. O'Reilly Media, Inc. ISBN: 9781449362942.

Cockburn, A., 2002. Agile Software Development. Boston, Addison-Wesley.

Dos Santos Rosado da Cruz A., M., R., 2010. Automatic Generation of User Interfaces from Rigorous Domain and Use Case Models. Ph-D Thesis, Departamento de Engenharia Informática, Faculdade de Engenharia da Universidade do Porto.

Ghiani, G., Manca, M., Paternò. F., Porta, C., 2014. Beyond Responsive Design: Context-Dependent Multimodal Augmentation of Web Applications. MobiWIS 2014: 71-85.

Kozuruba, S., 2010. Modellbasierte Anforderungs-analyse für die Entwicklung von adaptiven RIAs. Institut für Informatik Ludwig-Maximilians-Universität München, DiplomArbeit.

Limbourgh, Q., 2004. Multi-Path Development of User Interfaces. PhD-Thesis. Université catholique de Louvain.

Manca, M., Paternò. F., Santoro, C., Spano, L., D., 2013. Generation of Multi-Device Adaptive MultiModal Web Applications. MobiWIS 2013: 218-232.

Paternò, F., Santoro, C., Spano. L. D., 2009. MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. ACM Trans. Comput.-Hum. Interact., 16(4):1-30.

Puerta, A. R., 1997. A Model-Based Interface Development Environment, in IEEE Software 14(4), pp. 41–47.

Sossou, U., Shenoy, A., 2014. Learning Bootstrap. Packt Publishing. ISBN: 9781782161844.

Thevenin, D., 2001. Adaptation en Interaction Homme Machine: le cas de la Plasticité, Ph.D. thesis, Université Joseph Fourrier, Grenoble.

Valverde Giromé, F., 2010. OOWS 2.0: Un Método De Ingeniería Web Dirigido Por Modelos Para La Producción De Aplicaciones WEB 2.0. PhD thesis.