Development of a Design Model for Functionality and Content Access from Rich Internet Application Requirements

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

Keywords: Rich Internet Applications, User Interface Models, Metamodelling, Use Case Diagram.

Abstract: We have found several methodologies for the development of rich internet applications (RIA); however, they did not give enough attention to the problem of defining both appropriate notations and adequate process for developing the user interface (UI) of functionality and content access (UIFCA). The UIFCA is important, because it concerns with the global organization/behaviour of the UI of a RIA application; the UIFCA is complex in several RIA applications due to the several tasks/workflows/business process that need to be organized/accessed, and the use of single page applications and desktop like UIs. A good model for functionality and content access (MFCA) should be expressive enough, respect some abstraction requirements, and be understandable by the client; a good process to develop a MFCA should consider the creation of parts of the MFCA by the client, its completion by analysts, its early validation by clients, and the refinement of MFCA elements. In this work, we defined a metamodel called RIAFCA for building MFCAs, and a development process involving RIAFCA respecting these requirements. The metamodel and the process are illustrated with the help of an online e-mail application case study.

1 INTRODUCTION

There exist several RIA methods for the development of the application's UI or at least the UI of the application functionality (e.g. UWE (Kozuruba, 2010), WebML ((Brambilla, et al, 2010) and (Fraternali et al, 2010)), OOH4RIA (Melia et al, 2008), MARIA (Paternò et al, 2009), OOWS 2.0 (Valverde Giromé, 2010)); however, such methods did not give enough attention to the problem of how to define both appropriate notations and adequate method for developing the UI of functionality and content access. A UIFCA is of central importance (e.g., to clients and end users), because it concerns with the global organization/behaviour of the UI of the RIA application; in addition, a UIFCA is a complex part of the UI in several RIA applications due to the several tasks/workflows/business process that need to be organized/accessed, and the use of single page applications/desktop like UIs.

The *essence of a UIFCA* consists of the structure of it (i.e. how functions/content elements are grouped, and how groupings are organized) and of the dynamic change of the set of accessible functions/content elements to the user.

Notations of RIA methods found have limitations satisfying the following requirements that a good MFCA should satisfy:

R1. A MFCA should have a rich set of elements for describing the structure of a UIFCA, and to have a rich set of actions for modifying the accessible functionality/content elements of a UIFCA.

R2. A MFCA should abstract from the description of functionality, of UI elements for content output, of UI elements for data input and of access structures for inputs (e.g., menu, index, breadcrumbs).

R3. A MFCA should be understandable by clients (i.e. it must not involve elements corresponding to technical concepts not known by the clients,).

Usually RIA methodologies have abstract notations for describing the requirements and the UI, and concrete notations for describing the UI. In general, concrete UI notations are worse in satisfying requirements R2 and R3 than abstract UI notations, and usually abstract UI notations are not bad to capture the structure of a UIFCA. For these reasons, we consider as related work the abstract notations

for RIA requirements/UI. The abstract modelling notations found for RIAs have limitations on describing the essence of a UIFCA, they do not satisfy all requirements in R2 at the same time, and they are either not understandable by clients or very incomplete and understandable by clients.

The reasons for the above requirements are:

R2. 1) less aspects to think about when developing a MFCA; 2) after an early validation of a MFCA, the correction of errors in the MFCA will not obligate to make changes concerning the aspects abstracted by the MFCA; 3) less aspects to think about when changing a MFCA; 4) it is easier to consider changes to user requirements; 5) a MFCA remains stable when UI element descriptions for content elements are changed; 6) separation of MFCA description from: function description and the UI for output content/data input/access structures.

R3. This requirement allows the clients to: validate a MFCA, and to provide parts of such models (e.g., parts concerning innovative concepts and functions, parts that are not easy to comprehend by analysts).

RIA methodologies found have limitations w.r.t. the following requirements that a development process involving a MFCA should satisfy:

P1. The client is enabled to provide part of the structural part of the essence of a UIFCA.

P2. The analyst develops the part of the essence of a UIFCA not provided by the client.

P3. There is an early validation by the client of the essence of a UIFCA.

P4. There is a phase where content/input elements of a MFCA and requirements for the dynamic variation of the accessible functionality/content elements are refined; the refinements are expressed using UI elements of a UI notation that abstracts from layout, style and specific technological widgets, and is modality independent. This is to allow the mapping/adaptation of a MFCA with these refinements to obtain UIs considering different modalities/ devices/implementation technologies; in addition, if a content element is complex we can master the complexity of its development by first describing an abstract UI element for it, and next, incorporating modality, device, layout and style.

There is a lack of RIA methodologies considering P1, P3 and P4; with respect to P2, we have only found some RIA methods were the analyst develops the essence of a UIFCA with some limitations and without a participation of the client.

The objectives of this paper are: 1) the definition of a MFCA for RIAs satisfying the above requirements, and that is independent from modality and implementation technology, and 2) the definition of a development process satisfying the above requirements.

In Sec. 2 we defined a MOF metamodel (called RIAFCA) for describing the essence of a UIFCA, and respecting the abstraction requirements; to produce this metamodel we have taken some decisions in order to permit the client to understand its models. In Sec. 3 we defined a process considering: 1) the development of a RIAFCA model taking into account P1, P2 and P3, and 2) the refinement of RIAFCA model elements by using trace relationships for fulfilling P4.

2 RIAFCA METAMODEL

A *user role site view* is the part of a RIA UI used by a specific user role. RIAFCA abstracts from specific UI widgets and from specific devices. Each RIAFCA model contains some elements used to describe how the a role site view is organized into coarse grained elements (see Fig. 1); we define a concrete syntax for the this part of RIAFCA that looks like a screen with some regions and elements (for Access) inside - we assume that clients understand and may produce such kind of sketches.

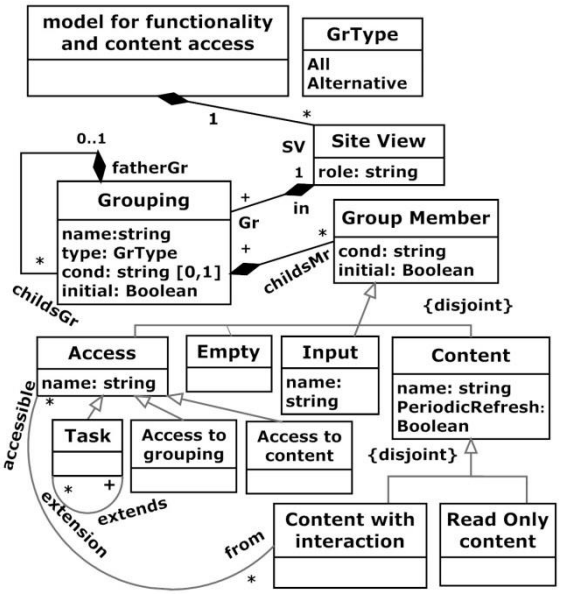

Figure 1: The RIAFCA part for describing the organization of the user role site views.

A *site view*: a user role site view. A *Grouping*: a piece of the UI for grouping Groupings or Group Members. Members of a Grouping can be either all present at the same time (*type=All*), or only one

present at a time (*type* = *Alternative*). A site view contains a hierarchy of Groupings and Group members. *A root grouping* is a Grouping at the root of its hierarchy. Grouping elements are represented with rectangles of different shapes according with the kind of grouping (See Fig. 2).

Figure 2: Concrete Syntax for groupings.

Input: - - for providing some input; *Access:* for accessing a functionality– use case, task; *Empty*: - ∅ - contains nothing; *Content*: for showing some content. *Content with Interaction* - - allows user interaction; *Read Only Content:* - - only for reading. *Task* - - for a task, a use case, a service, a command; *Access to grouping*: for navigating into a Grouping - -; *Access to conten*t: for navigating into a content; we use for access to read only content; we use for access to content with interaction. *PeriodicRefresh* set to *true* (use icon) means that the information of a Content element changes periodically. In the RIA UI abstract notations found only elements Empty and Content with interaction, classifications of content and of Access and association between content with interaction and Access are not present.

For choosing an Access inside a Content we use the meta-relation with roles *from* and *accessible.* For accessing from inside a task of extension tasks (that are not necessary for the extended task to exist) we use the meta-relation with roles *extends* and *extension*. When an alternative grouping *G* is not present, and is presented, we need to say that a child *E* of *G* is presented by default; to express this we put on *E initial = true*. We graphically represent a member *E* of *G* with *initial=true* with the rectangle of *E* filled with grey color.

Fig. 3 shows a *User Agent* grouping for a mail application. *Work*, *Commands*, *Lists,* are alternative Groupings. *Lists* Grouping contains 2 Content. *Commands* Grouping contains *Refresh* task, *Actions* Grouping and an *Empty*. There are 2 Access to grouping: *view settings* (to access a grouping for settings parameterization) and *Account group* (to access a Grouping with account information and tasks). There is a *view mail* access to content that is used to access the mail content in Work Grouping.

Some alternative Groupings have *conditions* on all of its members (use *cond* metavariables); such conditions are propositional formulas whose propositions have names of Groupings/Group members (a proposition is true if and only if its corresponding element is present). In the modeling notations for RIA we have found (i.e. abstract UI models, navigation models) the use of conditions for alternative groupings is not considered.

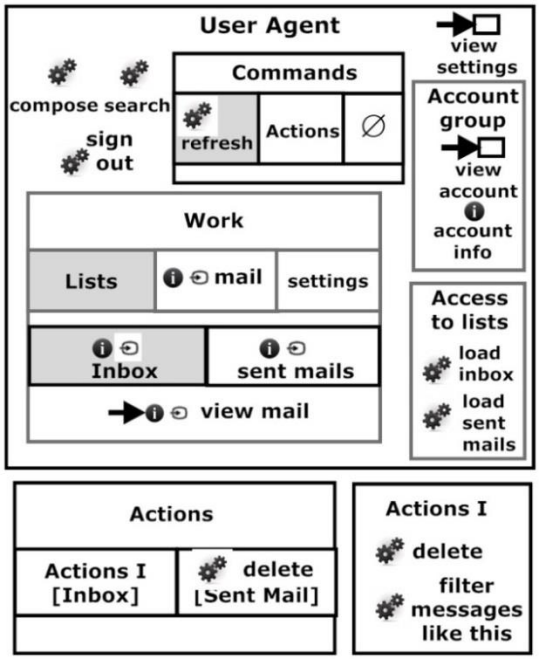

Figure 3: User Agent and Actions Groupings.

Fig. 3 shows the *Actions* alternative grouping; its member *Actions I* must be shown when *Inbox* content is present, and its member *Actions SM* must be shown when *Sent Mail* content is present.

To express requirements for the dynamic change of the accessible functions and content elements to the user, RIAFCA provides a set of modelling elements that are shown in Fig. 4; such elements are used to represent a set of *requirements* of the form: <user's interaction or another event, system's response>, where the *system's response* consists of one or more *actions* modifying the actual set of content elements and accessible functions.

The selection of this kind of notation was motivated by Pane and others (Pane et al, 2001) who conducted a pair of studies to examine the language and structure that children and adults used before they have been exposed to programming. In these studies, they presented programming tasks to nonprogrammers, who then had to solve them on

paper. In these studies they observed that an event-based or rule-based structure was used, where actions were taken in response to events.

Each action of the systems response of a requirement has a *Target* (i.e. Content, Grouping, Task, Empty) and a *type* that can be either open - -, remove - -, enable - -, disable - -, show - -, hide - -, interval - - (the target is presented only during a time interval). A *Requirement* says that after an event happened the actions on the targets must be performed; if an event has associated a Condition, the condition must be valid to perform the actions associated to the event; if an action has a Condition associated, the condition must be valid to perform the action. Only action types open, remove and interval are not present in the found abstract UI notations for RIAs.

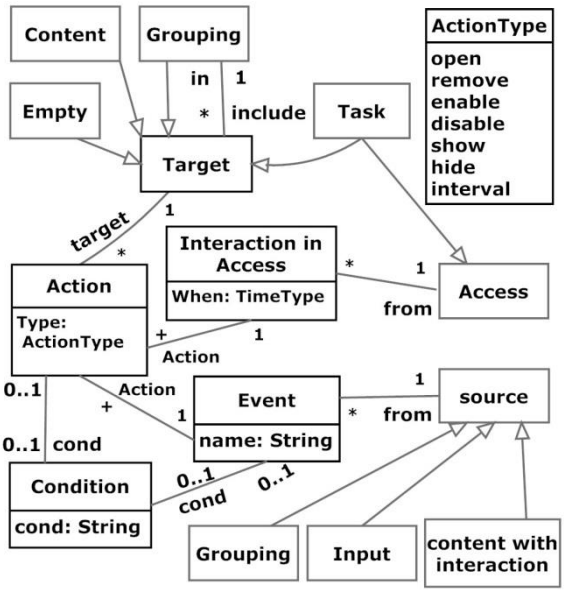

Figure 4: RIAFCA elements for expressing requirements.

Event elements can be: a) A user's interaction with a *source* element or another event associated to a grouping; for this case we use the icon ( ) together with the event's name. b) An Access is chosen (When=before and use), or an Access execution is finished (When=after and use).

A requirement is graphically represented with an arrow with one or more heads from the element where the event occurs to the Targets; the action type icons are put near the heads of the arrow. Event elements are shown on the tail of the requirement's arrow. A Condition is represented with the question mark (?) and a text for its description. A Condition associated to an event is put near the start of an arrow, and a Condition associated to an action is put near the head of an arrow.

Suppose that a target is a grouping *G*; if *G* is not associated with other targets (i.e. using the *include* association end), then *G* is presented with the default elements of its alternative groupings; else the targets associated with *G* are presented instead of the default elements of the corresponding alternative groupings. An associated Target with *G* is represented with an arrow with rhombus head from *G* to the associated Target.

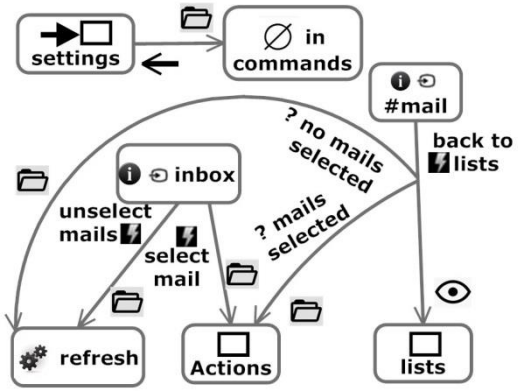

Figure 5: Some of the requirements associated with elements in User Agent grouping.

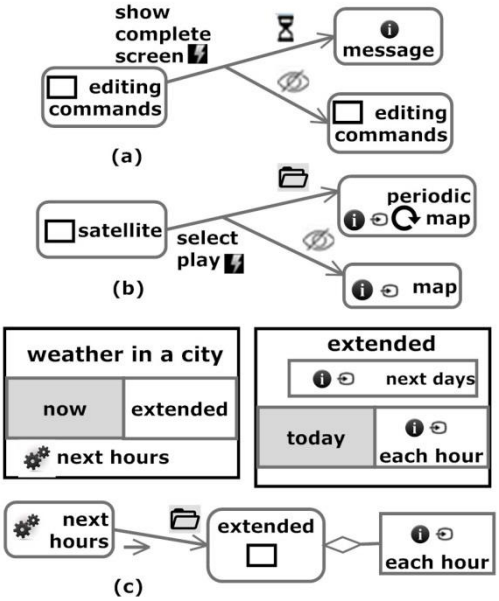

Figure 6: (a) show complete screen requirement, (b) play a weather forecast requirement, (c) view next hours requirement.

Fig. 5 shows some requirements associated with elements of *User Agent* Grouping (see Fig. 3).

When the user unselects all the mails in *inbox* the *refresh* task is presented, and when in *inbox* there are no mails selected and the user selects one, the *Actions* Grouping is opened. When the user choses to go back to lists in the *mail* Content, the *lists* Grouping is shown, the *refresh* Task is opened if there are no mails selected on the actual list, and the *Actions* Grouping is opened if there are mails selected on the actual list. The requirement at the top says that before presenting the *settings* grouping the *Empty* member is presented. Observe that the reading of the diagram should start with the initial elements (i.e. these elements without a # mark).

Fig. 6(a) shows a requirement for an online text editor. There is a window with both a *file* content and an *editing commands* grouping; when in *editing commands show complete screen* is selected, the *editing commands* grouping is hidden and a *message* content telling "press Esc key to view the editing commands menu" appears for an interval of time. Fig. 6(b) shows a requirement of a weather forecast application. There is a *satellite* grouping with a *map* content (satellite view of a region); when *play is selected*, the *map* content is hidden and *a periodic map* content with interaction with periodic refresh set to true is opened. Fig. 6(c) shows a requirement of a weather forecast application; this requirement says that after executing *next hours* task in *weather in a city* grouping the extended grouping containing an *each hour* content element is opened.

3 DEVELOPMENT PROCESS

First, the client develops some fragments of the RIAFCA model; next, the analyst develops some requirement models (e.g. use case diagrams, task models); following, the analyst using the fragments and requirements models, develops a complete RIAFCA model; next this model is validated by the client; using this feedback a revised RIAFCA model is constructed by the analyst; finally, elements of a RIAFCA model are refined into more concrete elements (e. g. UI elements on an abstract UI, events on a UI element).

Fragments of the RIAFCA Model Provided by the Client. This phase is to improve client´s satisfaction (we assume that an analyst lacks the domain knowledge that a client cannot easily convey when communicating requirements for a new application – such an assumption is a premise for End-User-Software-Development area – see (Paternò, 2013:1)).

The client could provide two kinds of RIAFCA fragments: 1) A decomposition of the root Grouping of a user role site view considering only the first levels of the decomposition; for each Grouping in this decomposition its purpose may be expressed. 2) Groupings for innovative concepts involving content and task elements related to the content (some of them may be accessible from the Content).

In Fig. 7 for the mail application the client provides an incomplete *user agent* Grouping, which is decomposed into: 1) *Commands* (for executing commands for lists of mails), 2) *Access to lists* (for choosing a list of mails to see), 3) *Work* (here the user may either interact with lists of mails, read mails, or configure the user agent), 4) *Account Group* (to manage the user account information).

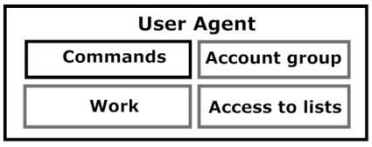

Figure 7: a skeleton of User Agent Grouping.

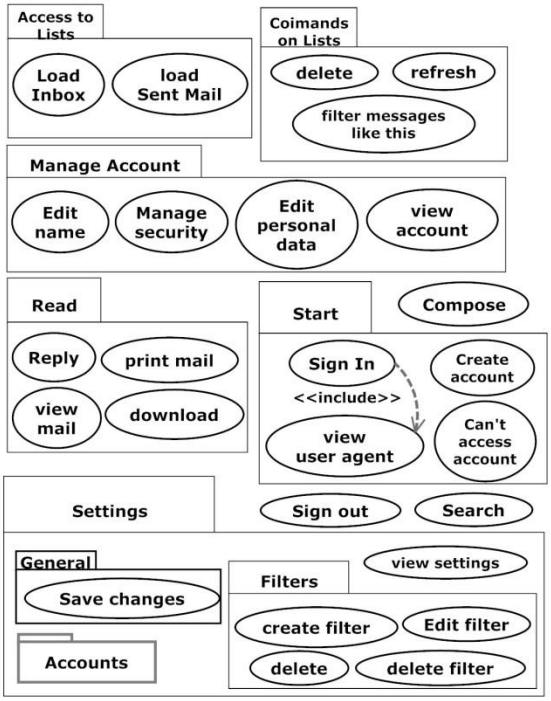

Figure 8: use case diagram for an E-mail application.

Requirements Provided by the Analyst. Examples of requirement models are use case diagrams (UCD), business process models, task models. We consider the case of UCDs from UML (see (Miles and Hamilton 2006)). Use cases (UC)

may be developed considering: a) Groupings for innovative concepts provided by the client. b) Other functional requirements provided by the client. In Fig. 8 you can see some of the UCs and UC packages for a mail application.

Development of a Complete RIAFCA Model by the Analyst. We assume that we have the fragments of a RIAFCA model provided by the client and a UCD available for the transition to a complete RIAFCA model; however, we do not prescribe a method for this phase. Independently of the method used, several decisions need to be taken by the method: **D1**: If a UC package *P* is mapped directly onto a grouping with the same name and containing mappings of its UCs and UC packages; in this case, the type of the *P* grouping is decided. **D2**: How to treat UC packages that are not mapped directly onto a grouping. **D3**: If a UC is mapped either onto a task or onto an access to grouping/ content. **D4**: Which are the content elements that are not provided by clients. **D5**: If the translation of a UC is accessible from a content or not. **D6**: Which are the UCs that affect a content element (i.e. modify, or process it). **D7**: Which are the members of the groupings of the first levels provided by clients. Depending on the method used these decisions will be made either manually or automatically or semi-automatically.

For the role site view *user* we create the *Mail* root grouping (See Fig. 9). We decided that Mail Grouping has two alternative children: *Start* (suggested by the *Start* UC package) and *User Agent*. For the *Start* UC package we considered D1as true; for the *sign in* UC we decided to introduce an input element called *Access data*. Next, we develop the *User Agent* Grouping of Fig. 3 from its skeleton; for *access to lists* UC package we considered D1 as true; from the purpose of the *work* Grouping we decided to decompose it into *lists* Grouping, *mail* Content and *settings* Grouping (D7). The *lists* Grouping is an alternative grouping with *inbox* and *sent mails* Content elements*;* they contain *view mail* Access to content. UCs *Compose*, *Sign out* and *Search* are mapped onto tasks that are put as children of *User Agent* Grouping (D3). We decided that the *Account group* Grouping contains an *account info* Content and a *view account* Access to grouping *Account*. The *Commands* Grouping contains tasks for the UCs of *Commands on Lists* UC package; however, for this package we consider D1 as false; the reasons are: a) for performing commands for lists, the *lists* Grouping must be present (in other case the *Empty* element must be presented), b) when no mails are selected only the *refresh* task may be executed; therefore, *Commands* must be an alternative Grouping containing *Empty*, *refresh* Task and *Actions* Grouping as alternatives (D2). For UCs of the *Read* UC package (with the exception of *view mail*) we considered D5 – i.e. their Tasks are put inside the *mail Content* box (See Fig. 9). The *Account* Grouping corresponds to *Manage Account* UC package. For the *settings* UC package we considered D1 as true, and the *settings* Grouping is alternative. For the *General* UC package we considered D1 as true, and a *general settings* Content is added. We needed a Content for the *actual filters*; for *create filters* and *delete* UCs affect *actual filters* (D6); from this Content UCs *edit filter* and *delete filter* are accessible (D5).

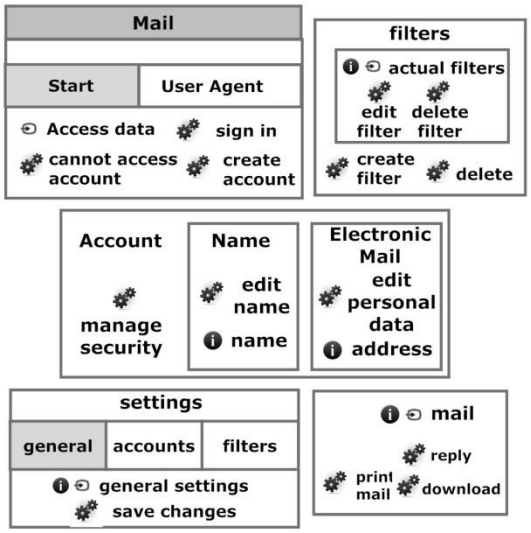

Figure 9: Other Groupings for the mail case study.

Next, the analyst expresses the requirements of a RIAFCA model; for each child of the root grouping of a user role site view a requirements diagram is developed.

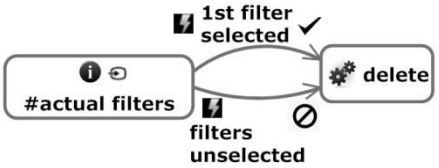

Figure 10: requirement associated to actual filters Content.

Examples: After the *sign out* Task is executed the *User Agent* Grouping is removed, and the *Start* Grouping is opened. In the *actual filters* Content the user may select or unselect filters; Fig. 10 says that when the first filter is selected, the *delete* Task is

enabled, and when all the filters are unselected, the *delete* Task is disabled.

Definition of Trace Relationships. The following tasks are considered: **T1**: If the UI model is legible by the client, then the client may provide UI elements (UIE) refining content elements (e.g. corresponding to innovative concepts). **T2**: Trace relationships between content/input elements and UIEs are constructed by using a UI model. We are not worried about how to obtain these trace relationships (e.g. automatically, manually). **T3**: Trace relationships between event/conditions in requirements and atomic events (possibly on UIEs)/ detailed conditions are constructed. We do not prescribe a method to obtain these trace relations.

We decided to use an abstract UI model for refining content and input elements that is platform and modality independent; this model must have a variety of content structures, access structures and basic UIEs. (Casalánguida and Durán, 2013) defines a UML profile containing design elements for RIAs called RIAAD considering such requirements. Now trace relationships between RIAFCA Content/Input elements and UIEs of RIAAD are considered; before explaining them, we include the definition of the needed RIAAD UI elements.

A *BasicUiElement* can be either an *Atomic* element or a *MediaObject*.. An *Atomic* can be: *text*, *number*, *anchor* and *selector* (i.e. *Single Choice* or *Multiple Choice*). *Atomic* elements have a *type of edition* attribute with values: *input* (for information input), *editable* (for information editing) and *no_editable* (for information presentation). *UIInput-Structure* represents a UIE used for user input; a special kind of UiInputStructure is a form. *Content-Structure* (CS) represents a UIE used for content presentation. Examples of CS are: List, Table, Tree, and Record. A CS can be *editable* (i.e. allowing the edition of some of its contents) or not. *Access-Structure* represents a UIE used for accessing other UIEs, or performing an action. Examples of AccessStructure are NavigationBar and NavList. *NavigationBar* represents a set of Anchors and one or more UiInputStructures. *NavList* represents a UIE containing a set of items; each item contains: optionally an anchor corresponding with content displayed for this item, optionally a navigationBar for parameters providing and/or functionality access, and BasicUIElements for describing an item.

Input Group Members can be refined into a UiInputStructure. Read only content elements can be refined into a no editable CS or a NavList. Content with interaction can be refined into a CS (e.g. an editable one) or into a navList involving possibly a navigationBar.

In Fig. 11 for the mail case study: Input *Access Data* is refined into a UiInputStructure with two text UIEs; *account info* is refined into a record with the same name with two text UIEs; *Inbox* Content is refined into a NavList with the same name containing items having an anchor to the mail, three text UIEs for mail information and a single choice UIE for mail selection.

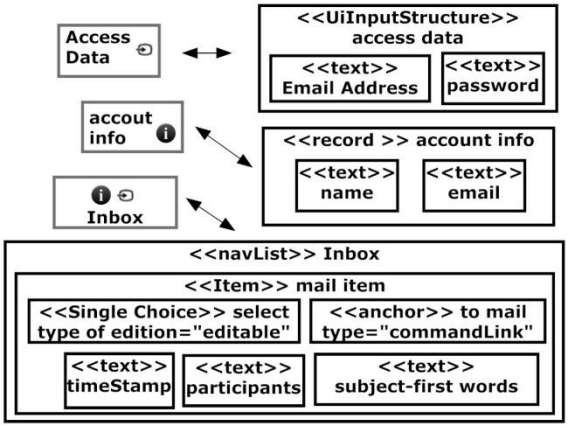

Figure 11: some refinements for mail application.

An *atomic event* consists of its name, its source and its data. We assume that in any given time of a web application execution, there exists a stream of the atomic events that happened; in addition, for each atomic event in the stream there is a time stamp for its occurrence.

We consider three kind of traces: traces from an interaction in Access element to an atomic event (perhaps on a UIE), traces from an event element to an atomic event (perhaps on a UIE) and traces from a Condition element to a more specific condition (perhaps referring to the UI).

Example: For the requirement in Fig. 5 saying to open the *refresh* task after all the mails are unselected on *Inbox* content, we have a trace from *unselected mails* to the event: *Select NO* on «single choice» *select* UIE inside «item» *mail item*.

Example: For the requirement in Fig. 5 with source the *mail* Content, we have a trace from *back to list* event to the event *Press* on «anchor» *back*, that is inside of «record» *mail* (from *mail* content there is a trace to a «record» *mail*); in addition, we have a trace from *no mails selected* Condition to "all *mail* items in actual list have their «single choice» *select* value equal to NO"; moreover, we have a trace from *mails selected* Condition to "some *mail*

items in actual list have their «Single Choice» *select* value = YES".

4 RELATED WORK

Tables 1 and 2 compare the relevant RIA approaches found in the literature. The references of these approaches are given once in the next paragraph.

Table 1: Comparison between abstract notations.

	R1	R2	R3
UWE Navigation M.	reg -	reg -	reg
UWE/R Navigation M.	reg -	reg -	reg
OOH4RIA Navigation M.	reg -	reg -	no
WebML Hypertext M.	reg	no	no
Rosado da Cruz UCD	reg	reg +	reg+
OOWS 2.0	reg	reg +	no
MARIA AUI model	reg	no	no
UsiXML AUI model	reg -	reg	reg ++
RIAFCA	good	good	yes

R1: *Captures the essence of the UIFCA:* for describing the dynamic change of the accessible functions/content elements: OOWS 2.0 (Valverde Giromé, 2010) interaction metamodel and MARIA (Paternò et al, 2009) dialog model have not open, remove and interval actions; (Rosado da Cruz, 2010) UCD notation has not open/remove, show/hide and interval actions; UWE/R (Filho and Ribeiro, 2009) considers requirements saying that after the execution of a task (e.g. a client process) some properties of the UI are changed (e.g. enable, disable, hide, show of an element of the UI, but not the other type of actions); navigation models of UWE (Kozuruba, 2010), WebML ((Brambilla, et al, 2010), (Fraternali et al, 2010)), OOH4RIA (Melia et al, 2008) have not action types; the UsiXML abstract UI model (Martínez Ruiz, 2007) does not consider the dynamic change of the accessible functions/ content elements. Concerning the structure of a UIFCA OOWS 2.0, UsiXML and the navigation models of UWE, UWE/R and OOH4RIA have not alternative groupings, UCDs in (Rosado da Cruz, 2010) have not content elements, and MARIA has not content with interaction elements.

R2: *Abstraction from description of output content, input element, access structures and functionality:* OOWS 2.0 RIA metamodel and UCDs in (Rosado da Cruz, 2010) do not abstract from functionality description. The other UI models for RIAs found do not abstract from functionality description. UsiXML does not abstract from input element description; UWE and UWE/R do not abstract from access structures; OOH4RIA does not abstract from output content description; WebML does not abstract from access structures and output content description; MARIA does not abstract from output content and input element description.

R3: *Understandable by the client, and the client may create parts of it:* notations that may be used by clients are: UCDs in (Rosado da Cruz, 2010) and UI abstract model of UsiXML - there is a concrete syntax based on sketches that is probably legible by the client to model part of the structure of the UIFCA. The navigation model of UWE captures part of the essence of the UIFCA, and we think that is understandable by clients if they know some concepts (e.g. index, menu, guided tour). UWE/R, OOH4RIA, WebML, MARIA have several technical concepts; OOWS2.0 has not a concrete syntax for the RIA metamodel, and the interaction metamodel has a rather complex textual syntax.

Table 2: Comparison between development processes.

	P1	P2	P3	P4
UWE	no	reg	no	no
UWE/R	no	no	no	no
OOH4RIA	no	no	no	no
WebML	no	no	no	no
UsiXML for RIAs	no	reg	no	no
Rosado da Cruz	no	reg	no	no
OOWS 2.0	no	reg	no	no
MARIA	no	reg	no	no
RIAFCA	yes	yes	yes	yes

P1: *The client is enabled to provide part of the structural part of the essence of a UIFCA.*

P2: *The analyst develops the part of the essence of the UIFCA not provided by the client:* in UWE the navigation model is generated from UML UCDs, and is refined; in OOWS 2.0 the RIA UI model is generated from an abstract interaction model, and the analyst produces an ECA model of the UI; in (Rosado da Cruz, 2010) the analyst produces an extended UCD; in UsiXML the AUI model is generated from a task model, and there is not a description of the dynamic variation of accessible functionality/content elements. In OOH-4RIA the designer produces the navigation model; in MARIA the abstract UI can be generated from a task model and additional information, and the generated abstract UI needs to be refined by the designer. WebML and UWE/R do not prescribe this task. The reason of rating as *reg* some methods is their limitations for modelling the essence of a UIFCA.

P3: *There is an early validation by the client of the essence of a UIFCA*: there is only a late

validation of a prototype in OOH4RIA, WebML (Rosado da Cruz, 2010), MARIA, and OOWS 2.0; this task is not prescribed by UWE/R and UsiXML.

P4: *Abstract UI elements (independent from modality, style and device) refining the content/ input elements of a MFCA are constructed.*

5 CONCLUSIONS

We considered the following case studies for identifying the elements of RIAFCA metamodel: an e-mail application, an e-commerce application, an online text editor, a weather forecast application.

For analysts/graphic designers to work with a RIAFCA with traces is better than to produce/use a UCD/navigation models/abstract UI model due to expressiveness of the RIAFCA, (see Sec. 4).

Our approach permits to deal with the complexity of a UIFCA: first construct a RIAFCA without worrying about UIEs; next construct the traces from RIAFCA elements to UIEs; finally, the graphic designers should only concentrate on widgets, style and layout.

RIAFCA metamodel abstracts from functionality description, from UIEs for describing content/input elements and from access structures; in addition, it is platform independent and modality independent. For these reasons, and because the RIAFCA considers ECA requirements, we think that analysts are in condition to develop RIAFCA models.

The reason for introducing our concrete syntax for RIAFCA requirements is to make this part of the RIAFCA understandable by clients, or at least very easy to learn by them.

For the mail case study we have 12 requirements from which 75% use open or remove actions, and are not replacing an element with another one. For the mail application for the transitioning from UCDs to RIAFCA static view we obtained: 55% of the UC packages are mapped directly onto Groupings, 33% of the UC packages needed to be distributed among more than one grouping, 11% of the UC packages are mapped onto a Content with Tasks inside. For the mail application 14% of the UCs are mapped onto Access to grouping/content elements.

For the future we plan to develop a tool that will consider: 1) the inspection of a RIAFCA model and of the trace relationships; 2) the generation of a program animating a RIAFCA where the client interacts with Content/Gouping/Access by clicking at event names inside of Groupings/Content or at Access elements, and looks at the resulting consequences; this is for permitting the client to understand even better a RIAFCA model.

REFERENCES

Brambilla, M., Fraternali, P., Molteni, E., 2010. A Tool for Model-driven Design of Rich Internet Applications based on AJAX. Handbook of Research on Web 2.0, 3.0, and X.0: Technologies, Business, and Social Apps., San Murugesan (ed.), pp. 96-118, IGI Global.

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, 13 th Intl. Conf. on Web Engineering*. Springer Verlag.

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

Filho, O., Ribeiro, J., 2009. UWE-R: An Extension to a Web Engineering Methodology for Rich Internet Applications. WSEAS Trans. Info. Sci. and App. 6(4): 601-610.

Fraternali, P., Comai, S., Bozzon, A., Toffetti Carughi, G., (2010): Engineering Rich Internet Applications with a Model-Driven Approach. ACM Transactions on the Web, Vol. 4(2).

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

Martínez Ruiz, F. J., 2007. A Development Method for User Interfaces of Rich Internet Applications. A Thesis for the Diploma of Extended Studies in Management Science. Catholic University of Leuven.

Melia, S., Gomez, J., Perez, S. and Diaz, O., 2008: A Model- Driven Development for GWT-Based Rich Internet Applications with OOH4RIA. In: *ICWE'10, 8th Intl. Conf. on Web Engineering*: pp.13-23, IEEE.

Miles, R., Hamilton, K., 2007. Learning UML 2.0. O'Reilly.

Pane, J., F. Ratanamahatana C. A., and Myers B. A., 2001: Studying the Language and Structure in Non-Programmers' Solutions to Programming Problems. *Intl. J. of Human-Computer Studies*, vol. 54, pp. 237-264.

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), November, pp 1-30.

Paternò, F., 2013. End User Development: Survey of an Emerging Field for Empowering People. ISRN Software Engineering, Vol. 2013, Article ID 532659.

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.