Designing the Interface of Rich Internet Applications

Matias Urbieta
LIFIA. Fac. Informatica, UNLP, Argentina.
matias.urbieta@lifia.info.unlp.edu.ar

Gustavo Rossi
LIFIA. Fac. Informatica, UNLP, Argentina and CONICET
Gustavo@lifia.info.unlp.edu.ar

Jeronimo Ginzburg
Dto. Computación, FCEyN, UBA, Argentina
jginzbur@dc.uba.ar

Daniel Schwabe
Dto Informatica, PUC-Rio, Brasil
dschwabe@inf.puc-rio.br

ABSTRACT

In this paper we present a novel approach for designing the interface of rich internet applications. Our approach uses the Abstract Data Views (ADV) design model which allows to express at a high level of abstraction the structure and behaviors of the user interface. Additionally, by using advanced techniques for separation of concerns it allows to create complex interfaces as oblivious compositions of simple interface atoms. Using a simple illustrative example we present the rationale of our approach, its core stages and how it is integrated into the Object Oriented Hypermedia Design Method (OOHDM). Some implementation issues are finally analyzed.

1. INTRODUCTION

Designing the interface of rich internet applications (RIA, from now on) [7] is difficult, as they must cleverly combine hypermedia-like interfaces of "conventional" Web software, with the kind of interface functionality we usually find in desktop applications (with drag and drop, pop-up information and diverse interface effects). To make matters worse, these applications must also deal with a myriad of concerns [10] which comprise multiple requirements both functional and non-functional and which usually crosscut each other.

The permanent "beta" state of rich internet applications complicate things further: new interface widgets or interaction styles are constantly introduced, checked to assess users' acceptance, and then either becoming core components or eliminated.

Consider for example part of the interface of Google mail as shown in Figure 1. We can see a mail core concern that provides functionality to deal with e-mails as similar mail clients. This application also exhibits services belonging to other concerns, e.g. the chat concern and the RSS concern. The chat concern allows users to send and receive instant messages by means of the browser and crosscut the mail concern by enhancing it with widgets like a semaphore showing mail contacts and by providing a "reply by chat to" button in the mail concern. Besides, some parts of the interface allow hypertext-navigation, such as the RSS feeds on top of Figure 1, which might be seamlessly composed with the other functionality. Sometimes navigation might proceed as in the "old" Web, but alternative styles of navigation are now possible such as transcluding the target page into the source.

The problem we face is how to clearly specify the behavior of the interface, in a way that we obtain a modular and abstract interface model which can be translated (automatically or manually) into a running implementation.

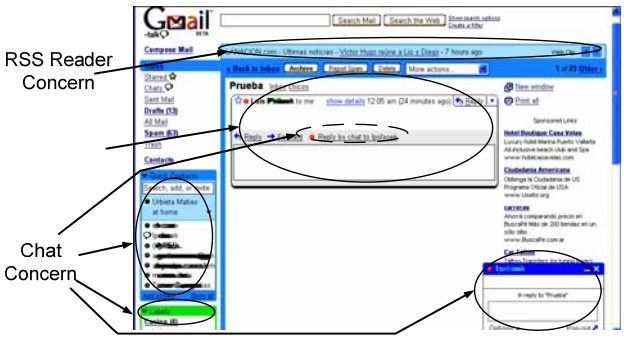


Figure 1. Different concerns in the interface of Google mail.

While there has been much research on development methodologies for "conventional" Web software [5,11,16], design approaches for RIA are just emerging [3, 14].

In this paper we present a systematic approach for designing RIA. The approach is a light-weight extension of the Object-Oriented Hypermedia Design Method (OOHDM) in which rich interface behaviors are specified

0-7695-3008-7/07 $25.00 © 2007 IEEE
DOI 10.1109/LA-Web.2007.14

by profiting from the object-oriented nature of Abstract Data Views (ADVs). The approach also uses some concepts borrowed from aspect-orientation to deal with cross-cutting compositions at the interface level. The paper has two important contributions: first it shows how to specify structural and behavioral transformations at the interface level using a single, uniform notation; second it presents an elegant way to composing crosscutting concerns with an extension of the ADV formalism. We show how to make the composition oblivious, i.e. such that components don't need to be modified or edited to be composed.

The rest of the paper is structured as follows: in Section 2 we briefly characterize RIA and give an overview of our approach; in Section 3 we describe the ADV design model and how we use it to specify RIA interfaces and in Section 4 we address the problem of complex interface compositions. In Section 5 we discuss some related work. Finally, in Section 6 we conclude the paper and present some further work we are pursuing.

2. Designing Rich Internet Applications

There are many features which characterize RIA from a Web Engineering point of view such as rich interaction capabilities, complex client-side processing, the elimination of full page refreshing to provide navigation, multimedia animations, etc. (See [3] for a complete characterization).

In the context of our research we are particularly interested in those applications which use rich interface behaviors to improve the usability of complex behavioral applications such as e-commerce sites, advanced Web mail clients (like Gmail or Yahoo mail), internet radios (such as Pandora), and generally so-called "Web 2.0" applications.. By using rich interaction features we can simplify the user's task, improve his access to information, make navigation more dynamic, etc.

For the sake of comprehension and conciseness we will not address "pure" multimedia applications, in which the access to application's behaviors are not the main target, even when these fancy applications could be also modeled using our approach.

A design approach dealing with applications which combine hypertext navigation with advanced interface behaviors should allow specifying:

a. The application or content objects which contain the basic information and behaviors of the application.

b. The hypertext nodes which the user will navigate and their relationships with application objects; additionally it should allow specifying different navigation semantics.

c. The interface look and feel of hypertext nodes and the interface of those application objects which are perceivable but not "navigable".

d. The way in which the application reacts to interface events: this includes the interface transformations which occur as a result of these events and how application behaviors are triggered.

Additionally, a modern design approach should support seamless composition of application and interface objects which correspond to different concerns, especially when these concerns crosscut. In this paper we will put our focus mainly on aspect d. above and give an overall idea of interface composition issues in Section 4.

In the next sub-section we describe the overall ideas of our approach; in section 3 we concentrate on the interface design stage.

2.1 Our Approach in a Nutshell

We closely follow the OOHDM design framework. Development proceeds in a five stage process comprising requirements modeling, conceptual modeling, navigation design, interface design and implementation. The first four steps generate an implementation-independent model which can be later mapped to different interface platforms (such as HTML, XUL, usiXML, etc**).**

In Figure 2 we see an overall schema of our design approach. Interface objects (the focus of this paper) are specified using the Abstract Data Views (ADV) approach described in detail in section 3; interface objects can be classified either as "pure" interface objects, therefore acting as behavioral controllers of other objects (e.g. buttons which trigger applications behaviors), as interfaces of navigation objects thus providing interface support for navigation (e.g. showing information or anchors of a node), or as interfaces of application objects; in this latter case interface objects trigger application behaviors not directly related with navigation. These three types of interface objects can be identified in the schema of Figure 2 according to their relationships with application or navigation objects.

As discussed in Section 4, interface objects corresponding to a single application concern (e.g. Mail, Chat, etc) are grouped together as components of a composite ADV. Finally ADVs of different concerns can be integrated either using traditional object composition or by means of aspect-like weaving [15]. Conceptual and Navigational classes can (and should be) also be grouped according to the corresponding concerns but we ignore this in the diagram for the sake of clarity.

In Figure 3 we show how this schema looks like for a part of the Gmail example. A simplified conceptual model

contains classes related to the two main application concerns: Mail and Chat. Mail messages have their navigational counterpart because it is possible to navigate through messages; for example we can traverse them as an OOHDM navigational context [16].

Chats meanwhile are not navigated; this means that we don't access them through links and we don't have anchors in chat elements. Therefore we don't model them as nodes (i.e. they don't have a navigational counterpart), though their interface must be specified, as shown in Figure 3.

We next describe how we design structural and behavioral interface aspects using ADVs; hypertext aspects are discussed in Section 3.3.

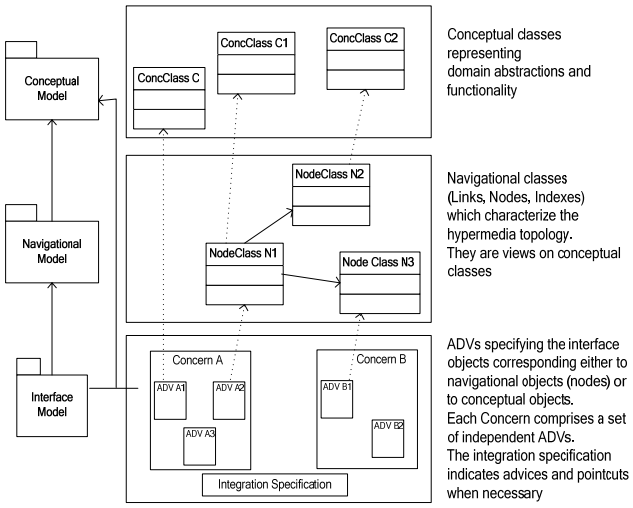


Figure 2: OOHDM design framework for RIA

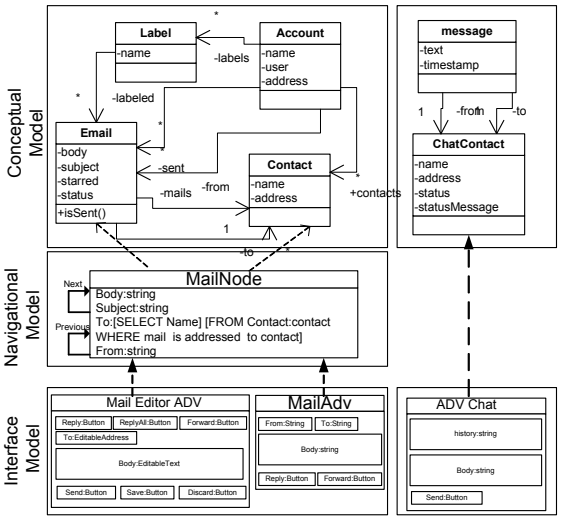


Figure 3: An instantiation of our schema for Gmail

3. Using ADVs to Design the Interface

ADVs [6] allow to specify the interface objects of a software application and how these objects relate with other application objects. Though the ADV approach was originally devised for conventional interactive software, it has been adapted and improved to be used in the context of Web applications [16], as ADVs can be easily mapped into running interface objects (e.g. XML/XSL specifications). An ADV is a composed object which possesses state and behavior; this behavior can be exercised by traditional method calls and also by interface or internally generated events (such as "mouse click"). ADVs can be composed or grouped in generalization/specialization hierarchies therefore allowing some level of reuse, when defining recurrent interface object types (like buttons, maps, etc.). They are abstract, as their specification is implementation-independent; however details about their lower-level aspects (such as location, background color, etc) can be annotated in the ADV specification or indicated as part of the ADV state. ADVs "observe" [9] application objects, known as ADV owners [6], in which the application data and business logic is usually managed. However, being full fledge objects they can contain arbitrary behaviors, including part of the business logic, which in conventional Web software is typically allocated in the controller of the Model View Controller triad (such as in J2EE tools like Struts or JSF). In RIA, meanwhile, this kind of business logic might be allocated in the interface [3].

An ADV specifies the interface aspects of its owner, i.e. how we intend the owner to be perceived by the user. In the context of RIA, and as shown in Figure 2, an ADV might be the interface of a navigation node or the interface of an application object (when no navigation is involved). ADVs may also relate with their owners not just to indicate the owner's look and feel but to trigger the owners' behaviors (which is the case with buttons, menus, list of options, etc).

The relationships with application objects are specified using configuration diagrams in which the exchange of messages between the interface and the core objects are shown [6]. Configuration diagrams are similar to UML class diagrams though they emphasize which messages clients send to servers (i.e. ADVs and their owners).

ADVs are also used to indicate how interaction will proceed and which interface effects take place as the result of user interaction. These behavioral aspects, which are specified using ADV-charts [4] (a kind of Statecharts), are of great importance for RIA. We describe below how to specify the structure of interface objects and then how to indicate their behaviors, for example to change their perception aspects (e.g. being visible or not), to start business logic, etc.

3.1 Structural Aspects

An ADV is described as an aggregation of lower level ADVs which reflect the composite structure of Web interfaces; to improve communication between stakeholders in the design process, we have slightly

modified the notation in order to indicate the relative spatial position of ADVs in the diagram, as shown in the examples.

In Figure 4, we show the ADV structure corresponding to a part of the Google mail interface of Figure 1 and a simple configuration diagram showing the relationships among the interface objects and their corresponding application objects.

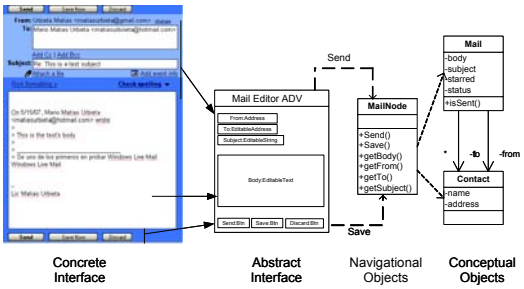


Figure 4: The static structure of a RIA interface

The Mail Editor ADV is one of the ADVs in the structure of the Mail concern; it is composed of some lower level ADVs, namely *From*, *To*, *Subject* and *Body*, the last three ones are editable fields which means that they will encompass some behavior to support editing. For the sake of conciseness we don't explain this behavior further, though it can be easily specified by treating the *Editable* behavior as a role which can be applied to strings or textual ADVs. The Mail Editor also possesses some buttons, whose behaviors allow interacting with the mail object (in this case instantiated from the MailNode class). The dashed lines from the Mail Editor to Mail Node indicate the messages that the editor can send to the Mail Node, namely *Send* and *Save*.

3.2 Behavioral Aspects

ADV-Charts are state machines that allow expressing interface transformations which occur as the result of user interaction. It has been shown that they are equivalent to StateCharts [4], though they are more expressive in communicating the dynamics of interfaces, as it is possible to nest states in objects and objects in states as shown in the examples of this section.

The composite nature of ADV-Charts allows (by nesting states into ADVs) indicating how different lower-level ADVs are affected when the user interacts with the system. They can be also used (in combination with configuration diagrams) to indicate the way in which conceptual or navigational operations are triggered by interface events. While the nesting of states in ADVs follows the Statecharts semantics, meaning that this ADV can be in those states (either AND-ed or XOR-ed), the nesting of ADVs inside states indicate which are the ADVs that might be perceivable in that state.

An ADV-chart transition is labeled with its name, the event and pre-condition which triggers it and its post-condition (usually an event which will trigger another change of state, a method call, etc.). In Figure 5 we show a simple example of in which we express a usual RIA behavior in the context of Gmail.

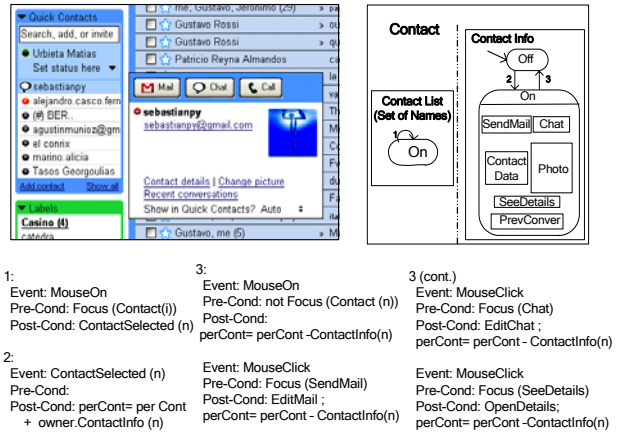


Figure 5: Specifying interface changes with ADV-charts

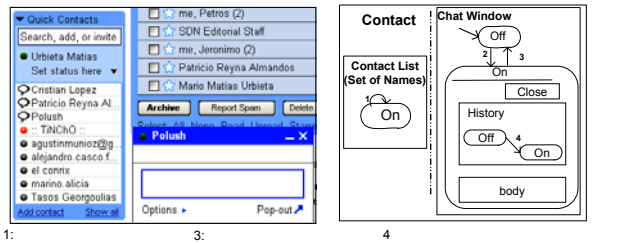
On the left of Figure 5 we show part of the actual interface and on the right a simplified ADV chart which indicates the interface behavior. When the mouse is on a contact person, the contact's data is shown.

The ADV *Contact* comprises two lower level ADVs: *Contact List* (left of the interface) and *Contact Info* (popped-up on the right), with an *And* relationship between them, indicated by a dashed lined between the ADVs. The *Contact List* ADV, which comprises a list of names, is always displayed (state "*On*" in the diagram). The *Contact Info* ADV is initially *Off*, indicated by the incoming arrow in *Contact Info*. Transitions are numbered, and for each one of them, we specify the event that triggers it, the pre-condition and the post-condition. We also use a function *Focus* which indicates the position of the cursor and a pseudo-variable *perCont* (referring to the perception context) to indicate the objects which are perceivable; these objects are "added" or "subtracted" from the perception context. ADVs also possess state variables which indicate their default position; this position can be also indicated in the post-condition specification as parameters of the operations on *perCont*. When the mouse is on a specific element of the contact list, *Contact Info* is made visible (transition 2). Notice that when the ADV is in state *On*, it shows some different sub-ADVs, in this case the *Send Mail*, *Chat*, *Photo*, etc.

Each one of these ADVs has its own interface behavior, indicated below the description of transition 3 (because all of them cause the transition to *Off*). These behaviors are described as post-conditions, which are themselves events to be interpreted in the context of the higher level ADV such as *EditMail*, *EditChat*, etc (See for example Figure 5).

The contents of the corresponding interface objects are obtained from its *owner* as shown in the specification of transition 2. For conciseness we omit the transitions which allow showing more contact details (triggered by the *OpenDetails* event) in the post-condition of transition 3.

A more complex ADV-chart showing some dynamics corresponding to the chat concern is shown in Figure 6. For the sake of comprehension we don't include the *ContactInfo* ADV shown in Figure 6, and which also belongs to the chat concern.



1:
Event: MouseClick
Pre-Cond: Focus (Contact(i))
Post-Cond: ContactSelected (n)

2:
Event: ContactSelected (n)
Pre-Cond:
Post-Cond: perCont= per Cont
+ owner.chatWith (n)

3:
Event: clickOn
Pre-Cond: Focus (close)
Post-Cond:
perCont= perCont –chatWith(n)

4
Event:
Pre-Cond: History owner.isNotEmpty()
Post-Cond:

Event: KeyPressed("Enter")
Pre-Cond: body.notEmpty
Post-Cond: perCont=
perCont
+HistoryText.append(bodyText)

Figure 6: An ADV showing the dynamics of the chat concern

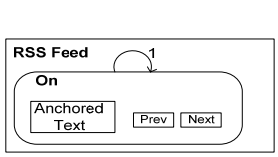
In the Figure 6, when the user clicks on an online contact the Chat window is opened containing the history of messages if it is not empty. This last behavior is implemented by the execution of transition 4, in the nested *History* ADV.

3.3 Hypertext Issues

As previously mentioned one of the most appealing features of RIA is that they allow combining the well-known style of hypertext navigation with dynamic interface behaviors; particularly, it is possible to make hypertext more dynamic (as in desktop hypertext environments).

In OOHDM we have used ADV-Charts to indicate the behavior of anchors [16] by expressing how the perception space changes when an anchor is selected. In the Web the usual response to this event is that the target node is opened and the source is closed, i.e. a new page is loaded instead of the former.

In Figure 7 we show the ADV-Chart which specifies a typical hypertext navigation operation from one of the news which appears on the RSS concern. In this case the source node (Gmail) remains open while the target node is opened in another window. Considering the semantics of Gmail for this specific link, the new window will be treated as an independent artifact, not related with Gmail.



1:
Event: MouseClick
Pre-Cond: Focus (AnchoredText)
Post-Cond: openBrowserWin (AnchoredText.url)

Event: MouseClick
Pre-Cond: Focus (Prev)
Post-Cond: AnchoredText.prevFeed

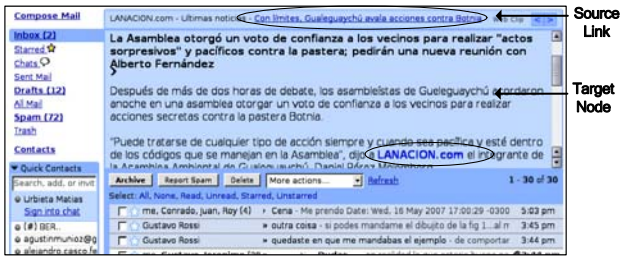
Event: MouseClick
Pre-Cond: Focus (Next)
Post-Cond: AnchoredTex getNewFeed

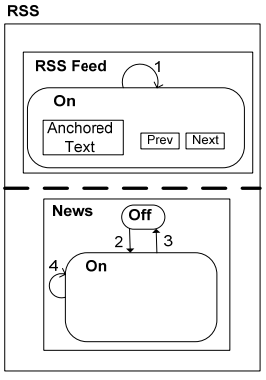
Figure 7: Specifying conventional navigation with ADV-Charts

This behavior is explained as the main consequence of transition 1 corresponding to the only state in which the RSS feed may be; the behaviors of the *Prev* and *Next* ADVs are specified below.

RIA however have the potential to exhibit much more sophisticated navigation semantics, though they require careful specification to insure consistent behaviors. An improvement of the behavior specified in Figure 7, which profits from the possibilities of RIA, would be to insert the target HTML document, corresponding to the selected news, in the same window where the link is shown, for example reducing the space devoted to the list of mails.

Figure 8 shows the application look and feel and the corresponding sketch of the ADV-chart which implements this behavior. Now the *RSS* ADV is an AND of the heading (described in Figure 7) and the *News* ADV; the main difference is that the "mouseClick" on the Anchored Text, causes the *News* ADV to be shown and the "reduceMailList" event to be broadcasted, so it will trigger a state change in the corresponding *MailList* ADV. Additionally if the Mouse is clicked outside the News area, the HTML text is eliminated from the perception context and the mail list is reestablished.





1:
Event: MouseClick
Pre-Cond: Focus (AnchoredText)
Post-Cond: openNews (AnchoredText.url)

Event: MouseClick
Pre-Cond: Focus (Prev)
Post-Cond: AnchoredText.prevFeed

Event: MouseClick
Pre-Cond: Focus (Next)
Post-Cond: AnchoredTex getNewFeed

2:
Event: openNews (url)
Post-Cond: News.content= url.getHTML;
perCont=perCont + News.content;
reduceMailList

3:
Event: MouseClick
Pre-Cond: Not Focus (News)
Post-Cond: perCont= perCont -News.content;
expandMailList

4:Event: MouseClick
Pre-Cond: Focus (NewsAnchor)

Figure 8: Improved Navigation in a RIA and corresponding ADV-Chart specification

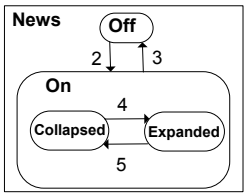
Using ADV-charts we can specify more sophisticated navigational behaviors; for example, notice that the text in the target node of Figure 8 has an outgoing link (marked in blue and surrounded with an oval). Following the RIA style for navigation described in Figure 8, clicking on that anchor might cause that the source node text is replaced by the target node (instead of opening a new window); this implies that the contents of the *News* ADV is now obtained from the target of that anchor. This behavior should be expressed in the Post-Condition of event 4, which was left unspecified in Figure 8.

We can even specify and implement a more "advanced" navigation style: transclusion of the target text into the source, in such a way that the target text can be later "closed" to return to the former state.

In Figure 9 we show how the *News* ADV-chart looks like when using a simple implementation of transclusion to navigate through news.

The *News* ADV-Chart now contains two sub-states indicating if the source text has been expanded with the target of the anchor's URL or "collapsed" (therefore only containing the source content).

The Post-Condition of transition 4 inserts the target text in the *News* ADV by sending the message insert to self. Meanwhile, after transition 5, the original text is compressed again, when the user clicks on the anchor again.



4:Event: MouseClick
Pre-Cond: Focus (Anchor)
Post-Cond: self insert ((anchor.url.content), anchor); perCon=perCon+ anchor.url.content

5:
Event: MouseClick
Pre-Cond: Focus (Anchor)
Post-Cond: self compress (anchor.url.content)
perCon= perCon – anchor.url.content

Figure 9: Specifying transclusion

As it is, the specification of Figure 9 only allows two states (Collapsed and Expanded) for the whole text. An alternative solution would be to consider that (when in state On), the *News* ADV has a list of anchors, each one with its corresponding (object) state and change transitions 4 and 5 to react according to the clicked anchor.

Transclusion can be even used at a pure interface level to implement one of the possible instantiations of the "Information on Demand" pattern [17].

In Figure 5 we showed how the contact information could be shown "on demand" when the user moves the mouse on the name of the contact. In Figure 10 meanwhile we show how we transcluded the map showing a company's address besides the mail body. In Figure 11 we show the corresponding ADV-chart specification. Notice that we are not navigating to a new node but just showing another attribute of the node which is being perceived with another style (map instead of text). The ADV corresponding to the map has also some owns behaviors which are not shown in the ADV-chart; they can be inherited from a more general ADV Map specification.

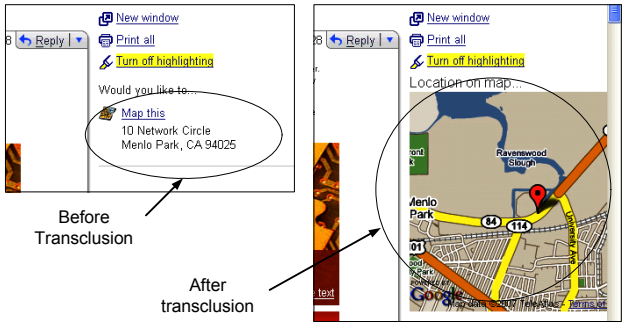
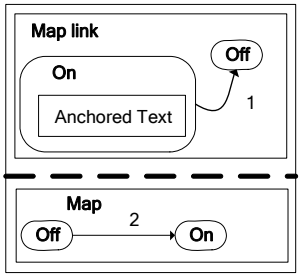


Figure 10: Transclusion at the interface level



1:
Event: MouseClick
Pre-Cond: Focus (AnchoredText)
Post-Cond: openMap (AnchoredText.url)

2:
Event: openMap (url)
Pre-cond:
Post-Cond: Map.content= url.getHTML;
perCont=perCont + Map.content;

Figure 11: Specifying transclusion in the user interface

4. Dealing with Cross-cutting concerns

As discussed earlier, a critical design issue for complex RIA arises from the fact that they deal with different application concerns, which in some occasions crosscut each other. While traditional composition mechanisms work well for integrating stable and/or orthogonal concerns, they have a drawback when recurrent editions of design models are necessary or when crosscutting is complex.

Some interface operations impact within the same concern in which they have been generated, such as operations in the mail concern in Gmail. Other operations might imply the execution of an operation in a different concern and therefore the concerns get tangled; for example when choosing "reply to Chat" in the mail ADV, the editChat ADV should be shown, etc. In this case tangling means that we have some interface objects or code of one concern (Chat) inserted in another concern (Mail); therefore, the evolution of one of them might impact in the other complicating maintenance.

To make this discussion concrete, we will present different examples of crosscutting features (structural and

behavioral) in the context of our Gmail example. All of them can be easily generalized and the corresponding solutions applied to other applications. Many kinds of Web applications might suffer this problem; however, given the dynamic nature of RIA interface behaviors, we mostly concentrate on those crosscuttings with are common to RIA.

Although, ideally, different concerns should be always clearly separated from requirements and through design to implementation, crosscutting might be particularly harmful when new concerns arise when the application evolves.

In Gmail for example, the Chat concern arose after the Mail concern was quite stable. As a consequence, adding this new concern implied dealing with those features of Mail that should be accommodated to support the new operations.

In [15] we presented an approach for weaving different ADVs by making them oblivious with each other and therefore allowing them to evolve separately. Basically, we propose to design the different ADVs independently such that there are no crossed references between them, and then integrate them by using a kind of specification very similar to the pointcut/advice style of aspect-orientation [8]. This specification is materialized using XSL transformations.

As an example in Figure 12 we show how the Chat and Mail ADVs can be integrated by indicating how the different Chat interface objects are weaved into the Mail Editor.

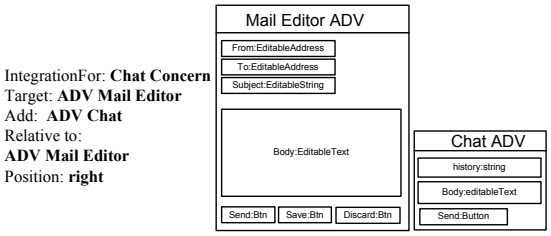


Figure 12: Weaving the Mail and Chat ADVs

This kind of structural crosscutting is of minor importance as it could be solved by treating both Mail and Chat as parts of a higher level ADV and therefore keeping them independently (instead of weaving Chat onto Mail).

Behavioral crosscutting however might be more complex to deal with. For example a new Gmail feature would advice the use of "Reply by Chat" feature in the Mail concern allowing to "switch" to the Chat mode when the recipient of the email is online.

In Figure 13 we show the actual interface and the corresponding confirmation ADV

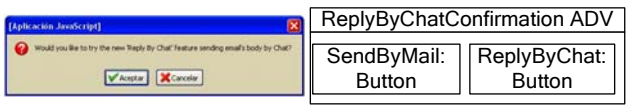


Figure 13: Confirmation ADV

To implement this functionality we need some way to intercept the original "Send" behavior such that the user is prompted to select the Mail or Chat mode. In the latter case the mail editor is closed and the mail content is sent by initiating a chat session. In Figure 14 we show the basic ADV-chart of the Mail concern in which the "MouseClick" event on the *SendButton* causes the email to be sent and the "MouseClick" event on *DiscardButton* discards the email. In both cases the event triggers not only a change of state but also a message is sent to the owner (the mail object).

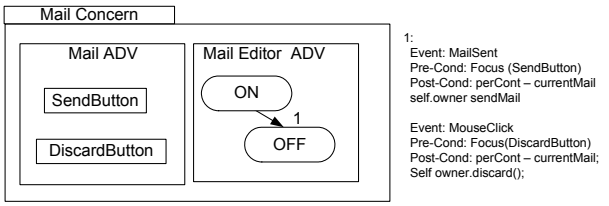


Figure 14: The basic Mail ADV-Chart

There are many different ways to incorporate the prompt of Figure 13 in the diagram of Figure 14; the first one is to ignore the impacts of crosscutting in evolution and hardcode the new behavior in the ADV-chart of Figure 14 by adding new ADVs and intermediate states.

A better solution is obtained by applying aspect-oriented concepts to ADV-Charts.

In [1] the authors suggest to use broadcast communication between AND-ed Statecharts to weave aspectual state transitions. In this case Chat is considered an aspect of Mail which provides the code for weaving (in this case the prompt). We don't comment this solution as it implies some (minimal) edition in the ADV-chart of Figure 14.

In [13], an improvement to this solution is presented by replacing the intrusive edition, through the specification of an event-reinterpretation to allow that when a transition in a core state-chart fires another in an aspectual one can be also fired. In summary, it is proposed to re-interpret transition 1 such that instead of its execution another one is fired.

We inspired ourselves with this last solution but considering that these application concerns are symmetric, i.e. Chat cannot be considered an aspect of Mail but must be treated also as a core concern. (A discussion on symmetry vs. asymmetry in aspect-orientation can be found in [8]).

Therefore, we propose that both concerns (and others) be developed separately and those features which correspond to their weaving (like the prompt in Figure 13) are specified separately as the structural weaving in Figure 12. And the new set of requirements will be designed asymmetrically in a new concern because they modify or call functionality within the former concern.

In Figure 15 we show the specification of the integration ADV-chart. It basically:

- Intercepts the MailSent event (using the "catch" keyword in the event 1)
- Enables the prompt in its state *On*
- Ends either returning control to the original ADV-chart (using the *proceed* keyword at the event 2) where the mail will be sent in the first fragment of transition 2 or
- It triggers the event which initiates the Chat and changes the effect of transition 1 in Figure 14 in such a way that the mail is not sent (second fragment of transition 2). This operation must send the email's body by chat and discard the real email, and therefore the caught MailSent event is dropped.

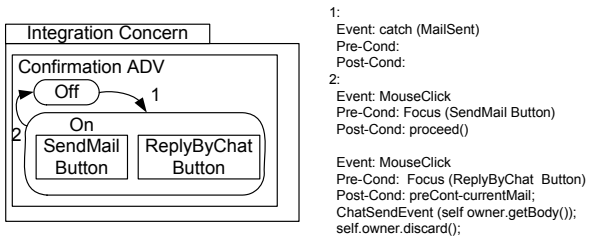


Figure 15: Confirmation Dialog's ADV-Chart specification

The impact of the introduction of the *catch* and *proceed* keywords in the mail concern is equivalent to appending a new state "Wait for Confirmation" and a new transition from the new state to off state. In Figure 16 we show the result of the weaving process in the Mail Editor's ADV-chart.

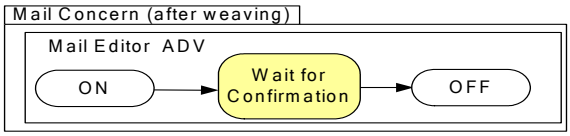


Figure 16: Final concern ADV-chart after weaving

4.1 Implementation Issues

Integration issues for crosscutting concerns should be dealt with implementation-neutral constructs as shown above. However, those constructs should be mapped to run time settings minimizing the impedance mismatch between them.

These behavioral graphical sketches can be implemented in a straightforward way in those GUI frameworks which supports event-driven actions such as HTML/ JavaScript, GWT, XUL, etc. Each event is translated to an action listener which is attached to a specified widget event, i.e. OnFocus, OnClick. Etc.

We have shown in [15] how to structurally weave ADVs by using XML transformations. Next we show a simple but effective way to implement interface weaving in a typical XML + Javascript based technology (like AJAX, Laszlo or XUL).

Figure 17 shows a sketch of a simplified Mail Editor ADV implementation in which we only show the most important interface objects related with the interface crosscutting.

In Figure 18 we show how the crosscutting behavior is implemented by using a JavaScript API (described in [2]) which provides aspect oriented features for JavaScript by taking advantages of JavaScript facilities to redefine functions at runtime and to wrap one function into another. The API provides functions that allow specifying function's point-cut where an advice can run before, after or around the joint-point.

```
<HTML>
<HEAD><!--head --></HEAD>
<BODY><!-- something -->
<FORM METHOD="POST" ACTION="sendEmailUrl">
        <!-- form's body -->
        <INPUT id="sendButton" TYPE="submit"
        value="submit" onclick="sendIt">
</FORM><!-- other things -->
</BODY>
</HTML>
```

Figure 17: HTML code of Mail Editor ADV

```
<!-- API inclusion -->
<script type="text/javascript" language="javascript" src="aspect2.js"
></script>
<script>
function ask(){
 if (confirm("Would you like to try the new 'Reply By Chat' feature
sending email's body by Chat?")){
        //send by chat
        return false;
        }else
        //go on as usually
        proceed();
}<!- - this code wraps the onclick function - ->
Aspects.addAround(ask,document.getElementById('sendButton'),
"onclick");
</script>
```

Figure 18: HTML code of the Crosscutting chat feature

The "ask" function in Figure 18 plays the advice's role which draws a confirmation dialog, and depending on the choice, it will send the message by chat or by mail (the original behavior). The function "Aspect.addArround" (in bold) is provided by the API and defines the point-cut that matches any "OnClick" event fired by the *sendButton* button with the aspect advice. This new block of code can be weaved obliviously onto the core interface code by an XSL transformation, like the ones we explain in [15]. The crosscutting code is just appended at the end of the Mail's interface code.

5. Related Work

The need for methodological support for RIA has been recently addressed [14]; a complete characterization of some design issues related with RIA has been given in [3], together with an extension of the WebML approach [5] to support RIA. More recently, in [12] a complete model-based approach to build interactive interfaces for RIA has been presented; the authors mention that this approach has been already implemented in the context of WebML though it is general enough to be "plugged" to other approaches. Our research follows a similar objective: the improvement of Web modeling and design methods for expressing the rich kind of behaviors that we find in RIA. Our approach is slightly different with respect to [3, 12], not only because it is based on OOHDM. Compared with [12] our approach only addresses the Abstract Interface Design stage and does not delve into architectural and implementation issues which, as in the rest of OOHDM research are treated neutrally. In this sense, the work in [12] completes the MDA life-cycle, while we have not developed implementation tools yet. We also address separation of concerns in user interface design, an aspect which has been so far ignored in the literature. In this sense, our work is equivalent to the proposal in [13] though it deals at the same time with structural and behavioral weaving, while aspect-oriented Statecharts in [13] only express behavioral crosscutting. Compared with the modeling approach described in [3] our work ignores lower-level aspects such as expressing which behavioral features should be dealt at the client or server sides; instead we have focused in the modeling primitives which are needed to express advanced interface behaviors like those shown in the paper.

6. Concluding Remarks

In this paper we have presented the most important aspects of our approach for designing interfaces of Rich Internet Applications. It is based on the OOHDM modeling and design framework and uses the Abstract Data Views (ADV) design model for specifying the structure and behavior of user interfaces of RIA. We have shown with simple but meaningful examples how to design the kind of interface transformations typical of RIA; by using ADV-charts (a variant of Statecharts) we are able to show which objects are to be shown or hidden in the interface, how information expands or collapses, etc. We have shown how to express different hypertext navigation semantics, including transclusion. Our approach encourages a clear separation of application concerns which can be later integrated using modern techniques for model weaving.

We are currently researching on several areas: first, we are studying how to map our design diagrams into implementation artifacts, maintaining the same modularity properties to assure graceful application evolution. The use of aspectual features in the user interface and the impact of interface crosscutting in the overall application's structure also deserve further research. Finally, we are analyzing how to integrate our approach with richer navigation design models such as StateWebCharts [18] to specify complex combinations between different navigation pages and their corresponding interfaces.

7. REFERENCES

1. Aldawud, O., A Bader and T. Elltad, Weaving with Statecharts, Aspect-Oriented Modeling with UML workshop at the 1 st *International Conference on Aspect-Oriented Software Development*, (2002)
2. Aspect Oriented Programming and Javascript. In http://www.dotvoid.com view.php?id=43 (2007)
3. Bozzon, A.; Comai,,S.; Fraternali,P; Toffetti Carughi, G. Conceptual Modeling and Code Generation for Rich Internet Applications. *ICWE2006*, Menlo Park, California, USA (2006)
4. Carneiro, L.M.F., Cowan, D.D and Lucena, C.J.P.,ADV Charts: a visual formalism for interactive systems.*SIGCHI Bulletin*, 26, 2, 74-77 (1994).
5. Ceri, S., Fraternali, P., Bongio, A. Web Modeling Language (WebML): A Modeling Language for Designing Web Sites. *Computer Networks and ISDN Systems*, 33(1-6), 137-157 June (2000)
6. Cowan, D. Pereira de Lucena, C.: Abstract Data Views: An Interface Specification Concept to Enhance Design for Reuse. *IEEE Trans. Software Eng*. 21(3): 229-243 (1995)
7. Driver, M; Valdes, R and Phifer, G.. Rich Internet Applications are the next evolution of the Web. *Technical Report*, Gartner (2005)
8. Filman, R., Elrad, T., Clarke, S., Aksit, M. (eds.). Aspect-Oriented Software Development. Addison-Wesley (2004)
9. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns. *Elements of reusable object-oriented software*, Addison Wesley (1995)
10. Gordillo, S; Rossi, G; Moreira, A.; Araujo, J.; Vairetti, C; Urbieta, M.: Modeling and Composing Navigational Concerns in Web Applications: Requirements and Design Issues. *In Proceedings of LA-Web* (2006)
11. Koch, N., Kraus, A., and Hennicker R.: The Authoring Process of UML-based Web Engineering Approach. In *Proceedings of the 1st International Workshop on Web-Oriented Software Construction* (IWWOST 02), Valencia, Spain, pp. 105-119 (2001)
12. Linaje, M; Preciado, J.C:; Sanchez-Figueroa, F.: A Method for Model-Based Design of Rich Internet Application Interactive User Interfaces. *Proceedings of ICWE 2007*, Como, Italy, July 2007, forthcoming.
13. Mahoney, M., Bader, A., Aldawud, O., Elrad, T., Using Aspects to Abstract and Modularize Statecharts. *The 5th Aspect-Oriented Modeling Workshop In Conjunction with UML 2004* (2004).
14. Preciado, J. C.; Linaje, M.; Sanchez, F., Comai, S.: Necessity of methodologies to model Rich Internet Applications, *IEEE Internet Symposium on Web Site Evolution*, pp 7-13 (2005)
15. Rossi, G., Ginzburg, J., Urbieta, M., Distante, D. "Transparent Interface Composition in Web Applications." *Proceedings of 7th International Conference on Web*

Engineering (ICWE2007: July 16-20, 2007; Como, Italy), pp. 152-166 (2007)
16. Schwabe, D., Rossi, G.: An object-oriented approach to web-based application design. Theory and Practice of Object Systems (TAPOS), *Special Issue on the Internet*, v. 4#4, 207-225. October (1998)
17. Schwabe, D., Rossi, G.: Improving Web Information Systems with Navigational Patterns. *Computer Networks* (31), pp 11-16, 1999.
18. Winckler, M., Palanque, P. StateWebCharts: a Formal Description Technique Dedicated to Navigation Modelling of Web Applications. *International Work-shop on Design, Specification and Verification of In-teractive Systems* (DSVIS'2003), Funchal, PT. (2003).