End-user Friendly UI Modelling Language for Creation and Supporting Evolution of RIA

Chris D'Souza[1], Athula Ginige[2] and Danny Liang[2]
[1]*School of Business, Australian Catholic University, 40 Edward Street, North Sydney, Australia*
[2]*School of Computing, Engineering and Mathematics, University of Western Sydney, Sydney, Australia*

Keywords: Rich Internet Application (RIA), End-user Modelling Language, Evolution.

Abstract: End users have a comprehensive understanding of business needs which is often hard to fully capture. One possible solution to this is empowering end-users to create and manage business applications. To empower end-users the paper presents an end-user friendly UI modelling language. The language facilitates the creation and supports the evolution of RIAs with changing business needs. The modelling language is based on various types of structural dependencies among the interface elements in RIAs. These structural relationships are identified in the paper. It also derives the data model from the end-user UI specifications. Evolution is discussed from three perspectives, namely, the structural model of interfaces, the behavioural model of interfaces and the underlying data model.

1 INTRODUCTION

Businesses requirements change frequently and consequently business applications need to evolve. End-user development is one way to manage frequent changes. Since many end-users perceive applications through UIs, they may be empowered by letting them specify the structure and the behaviour of the application through UI elements.

The paper presents an end-user friendly UI modelling language to facilitate the creation and evolution of a class of current web applications called Rich Internet Applications (RIAs). RIAs are web applications with desktop like user interfaces and response times (Busch and Koch, 2009). Several web engineering methods have been proposed to model RIAs. The important ones are OOWS (Valverde et al., 2009), OOH4RIA (Garrigós et al., 2009; Melia et al., 2010), UWE-R (Busch and Koch, 2009), WebML Extension (Bozzon et al., 2006) and OOHDM Extension for RIA (Urbieta et al., 2007). These methods are helpful to IS experts such as designers and developers. However end-users do not possess the skills required to intricately model the requirements using sophisticated modelling methods though they have an expert understanding of new and existing requirements. Further these methods do not facilitate evolution (Liang and Ginige, 2007). Several technological frameworks also exist that enable developers to expedite the RIA development process. Two prominent examples include Microsoft's Silverlight (Silverlight, 2010) and JavaFX based on Java technology (JavaFx, 2007). However these frameworks are platform specific and hence not transformable from one model to another. Some UI modelling languages also exist. E.g. the USer Interface eXtensible Markup Language (USIXML) uses a User Interface Description Language (UIDL) allowing designers to apply development of user interfaces at various levels of abstractions (Limbourg et al., 2005). Similarly the User Interface Markup Language (UIML) provides support to designers in modelling the structure, style, display content and the behaviour of the UI elements in multiple computing platforms (Farooq Ali et al., 2005). The eXtensible Interface Markup Language (XIML) is another UI modelling language with design time and runtime support for designers (Puerta and Eisenstein, 2003). These UI modelling languages support model driven engineering approaches. However most of these UI description languages are meant for designers. Hence an end-user friendly UI modelling language is proposed to fill the gap that exists between UI modelling languages for designers and end-users.

An overview of the suggested approach for developing RIAs is as follows. Expert end-users use a GUI based Integrated Developmental Environment

D'Souza C., Ginige A. and Liang D..
End-user Friendly UI Modelling Language for Creation and Supporting Evolution of RIA.
DOI: 10.5220/0004078201900198
In *Proceedings of the 7th International Conference on Software Paradigm Trends* (ICSOFT-2012), pages 190-198
ISBN: 978-989-8565-19-8

to generate an end-user friendly textual model of the expected UI in the form of one or more pages in the application. The textual model of the UIs is then processed by an engine to derive abstract and concrete UI models for each page along with the underlying data model. However the scope of this paper is restricted to the textual representation of the UI model of a single page and its effect on the data model, leaving aside its engineering details and validations for future work.

The paper is organized as follows: Section 2 contains a discussion on related work on RIA UI modelling and UI modelling lanaguages in general. Section 3 briefly discusses the research methodology. Section 4 classifies structural relationships among UI elements. Section 5 discusses the developmnt of the end-user friendly UI modelling lanaguage models based on structural relationships among UI elements. Section 6 contains the conclusion and future work.

2 RELATED WORK

Valverde and Pastor (2009) provide a specification of RIA UI meta-model as a combination of static views and dynamic views. The static view identifies the fundamental UI element types in a web application while the dynamic view identifies the fundamental behavioural changes to the UI due to user interaction. They then integrate the meta-model with the OOWS method to engineer the web application. Sections 2.1 and 2.2 provide an overview of Valverde and Pastor's RIA UI meta-model. Section 2.3 discusses current UI modelling languages while Section 2.4 discusses end-user empowerment through end-user friendly modelling language.

2.1 Static View of the UI Meta-Model

The UI can be defined as a composition of widgets (Valverde and Pastor, 2009). A widget is a visual component of the UI. Its main responsibility is to handle data and user interactions. A widget is abstracted as an entity with a set of properties. Five types of widgets are identified based on their interactive functionality:

Data View Widget: This widget displays data.
Input Widget: Allows user to input data.
Navigation Widget: The navigation widget captures the target from which the UI is perceived.
Service Widget: Service widgets initiate the execution of a service from the business logic.
Layout Widget: This widget contains other widgets.

2.2 Dynamic View of the UI Meta-Model

When a user interacts with the widgets, events are triggered which causes reactions on either the same widget or on other widgets (Valverde and Pastor, 2009). The reactions are in the form of:

Property Change: This reaction results in a change of the UI properties of any target widget.
Data Request on Demand: This reaction results in a request for information from the server to a data view, if it is not already available on the client side.
Functional Invocation: This results when a service widget triggers an event resulting in a requests-response communication with the business logic.
Input Validation: This reaction results in a validation of input data and a message if there is a problem with input data.
Navigation: The navigation reaction results in changing the point from which the application's UI is perceived by the user due to an event triggered from a navigation widget.

In addition, the dynamic view uses event rules to define reactions on target widgets for each event from a source widget.

2.3 UI Modelling Languages

UI modelling languages are generally employed to enable designers to generate UIs from various models such as domain, presentation and task. The generated UIs can then be customized by the designer to expedite the UI development.

Teallach, for example enables designers in building a UI from task, domain and presentation models at logical and physical levels and also maps the concepts from one model to another (Griffiths et al., 2001). USIXML is another UI description language that expresses and manipulates UIs at different levels of abstractions (Limbourg et al., 2005). These levels include Task & Concept (T&C), abstract UI (AUI), concrete UI (CUI) and Final UI (FUI) level. The T&C level describes common end-user interaction task objects in a given domain. The AUI level defines interaction space objects by grouping task objects according to requirements but without considering the specificities of layout and navigational elements. The CUI level defines objects from the AUI level with layout and navigation specifications but without considering the platform in which the rendering occurs. The FUI level defines

the CUI objects with respect to a specific computing platform. All UIDLs aim to provide designers a mechanism to bridge the time gap that exists between the user-interface engineering tasks of design, development, and evaluation. To reduce the time gap, UIDLs derive the UIs from domain model and task models. However end-users are not experts in these models. End-users on the other hand may be empowered by providing a UI language from which domain and other models could be derived.

2.4 Empowering End-users through End-user Friendly Modelling Language

Ko et al. (2011) observed that not much research has been done on end-user modelling of requirements and specification for interactive and web-based applications. Providing natural language like descriptions of requirements is one approach to enable end-user modelling, where in domain level keywords are mixed with the user defined terms in the language (Liu and Lieberman, 2005, Little and Miller, 2006). Liang and Ginige (2007) use a Smart Business Object Modelling Language (SBOML) which uses succinct, pseudo-English sentences to model relations among business objects. E.g. the SBOML statement "`in organisation, employee has first name, last name might have many office (has room number, building id)`" is user friendly as it is easily understood by end users. SBOML develops a platform specific model of a web application from its SBOML specification and supports rendering of the UI based on default mappings between data elements and UI elements. Though SBOML is not a UI modelling language it demonstrates that web applications can be built by empowering end-users to exploit their requirements' expertise.

3 RESEARCH METHODOLOGY

The research methodology is based on the ontological models identified in the Unifying Reference Framework (Calvary et al., 2003). The Unifying Reference Framework is commonly used for the design time and run time adaptations of UIs targeting multiple platforms. The ontological model in the Unifying Reference Framework is independent of any domain and interactive system and has been applied in UI description languages such as USIXML (e.g. see Limbourg et al., 2005). The framework recommends abstracting the UI models in separate layers. The separation of levels is essential for model driven engineering, reuse and the evolution of applications.

To empower end-users, an end-user UI modelling level is proposed to be situated over Valverde and Pastor's (2009) meta-model of the RIA UI. Hence the proposed RIA development process is as shown in Table.

Table 1: Proposed RIA UI Developmental Process.

End-User Friendly UI Model ↓
Valverde & Pastor's (2009) RIA UI Model ↓
Concrete Model ↓
Final Implementation

A UI modelling language may be termed user friendly if an application can be described with a UI perspective that is familiar to the end-user, enables tweaking of the application so that it fits one's personal needs and enables automation of repeated tasks(Cypher, 1993). These principles can be used during the evaluation phase of UI engineering.

4 STRUCTURAL RELATIONSHIPS AMONG UI ELEMENTS

This section identifies the structural relationships among widgets in UIs. It is important to identify the various structural relationships among widgets in the UI model because when the application evolves the relationships need to be redefined to support the new behavioural expectations. The following types of structural relationships are identified:

4.1 No Relationship

A widget has no relationship with any other widget except its container widget if no other widget's existence is dependent on it or if its own existence is not dependent on any other widget.

4.2 Container Relationship

Two widgets are in a container relationship if one widget is contained within the other widget. Hence two types of widgets may be defined:

Container: A widget in a container relationship is the container if it contains the other widget. Every web page has at least one container widget. A container widget may have container relationships

with other widgets.
Widget: This represents any widget in a container.

4.3 Computational Relationship

Two widgets are in a computational relationship if they are related by a computational expression. The relationship results in two types of widgets:
Computationally dependent: A widget in a computational relationship is computationally dependent if it uses a computational expression.
Computationally independent: A widget in a computational relationship is computationally independent if does not use a computational expression.

Consider a text field that displays the "total years of experience" of an employee by computing the value from two columns, "starting year" and "ending year", of an "experiences" widget (see Figure 1). Here "total years of experience" is computationally dependent on "experiences", the computationally independent widget.

4.4 Logical Relationship

A widget is in logical relationship if its existence is dependent on a logical expression. Two types of widgets may be defined:

Logically dependent: A widget in a logical relationship is logically dependent if its existence is dependent on a logical expression being evaluated to be true.
Logically independent: A widget in a logical relationship is logically independent if it exists independently of the value of the logical expression.

Consider a text widget "Number of children" that appears when the value of a "marital status" radio button widget is set to "married". Here "Number of children" is logically dependent on "Martial status", the logically independent widget.

4.5 Event Relationship

A widget is in an event relationship with another if it is set to raise an event that causes reactions on the other widget. Two types of widgets may be defined based on event relationships:

Event generating widget: A widget in an event relationship that generates the event.
Event target widget: A widget in an event relationship on which a reaction is applied.

By default all navigation widgets are event generating types. Furthermore, all input widgets are event generating types though not all of them may require the events to be handled. In addition every web page is assumed to generate an "on page load event" to cause the loading of data on the page.

4.6 Pop-up Relationship

A widget is in a pop-up relationship with another widget if the other widget appears as a pop-up page. Two types of widgets are defined based on the pop-up relationship:

Pop-up generating widget: A pop-up event generating widget.
Pop-up page: An event target widget that manifests as a page that pops-up on the occurrence of the event. Pop-ups are frequently used to create contextual menus, confirmation dialog boxes or validation messages.

5 DEVELOPING AN END-USER FRIENDLY UI MODELLING LANAGUAGE

An approach to end-user empowerment is to create applications from a UI perspective, using an end-user friendly modelling language. The modelling language must also be capable of supporting the evolution of an application in the form of addition, deletion and editing of widgets in the UI. The UI model will also have an impact on the underlying data model and the behavioural model. These issues are discussed next.

5.1 Contextualizing UI Relationships

The UI of an application is represented in the form of one or more pages. A page may contain at least one container widget and a container widget may have container relationships with other container widgets. Furthermore widgets in computational relationship, logical relationship, event relationship and pop-up relationship can be widgets in a container. Finally a pop-up target widget may be perceived as a page. The structural representation of the relationships among the UIs in an application may be stored at the abstract and concrete level using XML notation so that they can be easily transformed from one model form to another.

5.2 Language Support for Creation

Consider the UI of an application to manage employee details (see Figure 1). The UI has data

view widgets for “F.Name”, “Experiences” table and “Total Years of Experiences”. Assume “Number of kids” is a logically dependent widget on “Marital status”. “Number of kids” is an input cum data view widget. Also assume “Total Years of Experiences” is computationally dependent on the “Experiences” table using a computational relationship involving data from “Starting Year” and “Ending Year” columns of the “Experiences” widget to calculate the total years of experience. When “Click here for experiences” widget is clicked, it opens a pop-up page.

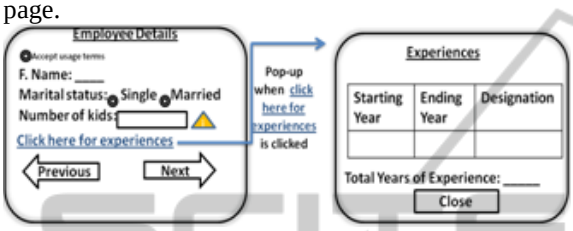

Figure 1: UI of an application.

A page may contain additional widgets to supply “information” or to receive “confirmation”. Information widgets are data-view widgets providing helpful information about other widgets. E.g. the “Number of Kids” widget has an information widget represented by the yellow triangle to inform the user to enter a positive integer in the “number of kids” widget. A confirmation widget is a data-input widget generally used to gather confirmation of terms and conditions before proceeding with the navigation.

5.2.1 Script for Creation

Script 1 defines the end-user model of the application illustrated in Figure 1. The modelling language has a small set of keywords to sufficiently define the structural relationships identified in Section 4 and its consequential mapping to UI widgets discussed in Section 2.1.

The script defines an application using one or more statements. A statement ends with a semi-colon. Flower brackets in a statement represent a container widget. Properties of a widget are represented within round brackets. Multiple properties are separated by a comma. A property may have sub-properties which are represented within nested round brackets.

Script 1: Modelling creation of the UI

```
01 In application <application_id>
02 create page<Employee Details>
03 (on load event,
04
05 for each <Employee>
06 {
07 (
08 <Accept usage terms>((Uni-choice:
09  <Accept usage terms>), hide
10  label, confirmation widget);
11 <F. Name> (String, read);
12 <Marital status>
13 (Uni-choice: (<Single> (checked),
14    <Married>), on set event);
15
16 <Number of kids>
17  (String, read-write,
18  logically dependent on
19  <Marital status> set equal to
20  <”Married”>, info widget
21  (<Number of kids must be more
22   than or equal to 0>));
23 <Click here for experiences>
24  (link, on click event, to pop-up
25   page<Experiences>
26  {
27  <Experiences> (table(
28   <Starting year> (integer);
29   <Ending year> (integer);
30   <Designation> (String)
31  ));
32
33  <Total Years of experience>
34   (integer,
35   computationally depended on
36   (sum (subtract
37   (<Experiences: Ending year>,
38    <Experiences: Starting year>
39   ))))
40  })
41 )
42 }
43 )
```

In the script, underlined text represents keywords of the language. Line 1 identifies the application. Line 2 identifies the page. Lines 3 to 43 describe the properties of the page. Line 3 states that the page is set to have an on load event. Line 5 uses keywords to describe a widget which is identified as a container in line 6. The keywords "for each" convey additional information to generate widgets for navigation from one employee detail to another. Lines 7 to 41 describe the properties of the container. They describe five widgets, namely “Accept usage terms”, "F. Name", "Marital status", “Number of kids" and "Click here for experiences". Line 10 identifies “Accept usage terms” as a

confirmation type widget. Line 20 indicates "Number of kids" has an information widget attached to it. Lines 25 to 40 describe the "Experiences" container" and it's associated widgets. The container is also defined to be in a pop-up page in line 24.

The language can also capture optional data type and the read/write properties of widgets, See line 11 for an example of a string data element with a read property and line 8 for a uni-choice data type to represent a choice among one or many alternatives. The language also supports navigation widget properties. E.g. Line 24 uses keyword "link" to define a hyperlink type of navigational widget.

5.2.2 Deriving the Structural UI Model

The structural relationships among the widgets in the UI can be automatically derived from the language. Figure 2 represents the structural object model of the UI defined by Script 1. Though Figure 2 is shown in UML notation, it may be represented in XML form for easy storage and manipulation to other levels of abstractions.

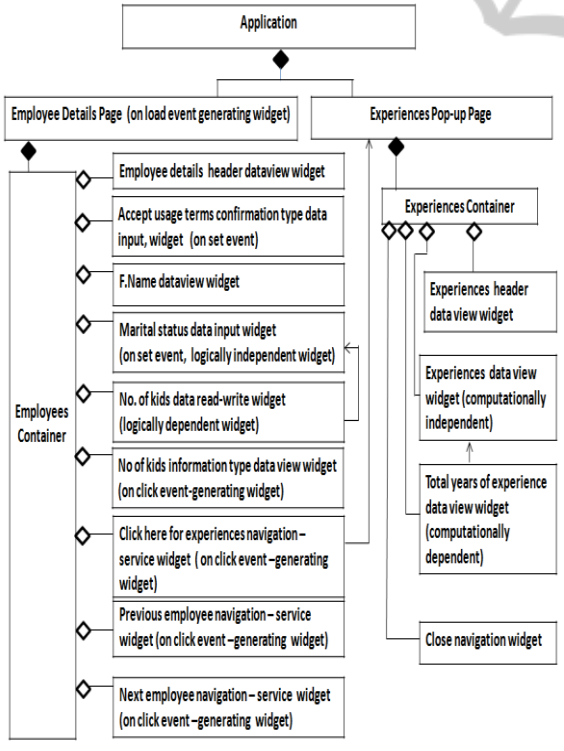

Figure 2: Structural object model of the UI.

The structural object model may contain additional widgets not explicitly defined by the end-user. E.g. the "Previous employee navigation-service widget" is not user defined. This is generated due to the keywords "for each" on line 5 in Script 1 which is mapped to two buttons "previous' and "next" in Figure 1. Similarly the structural object model may define other implicit widgets such as a default header widget for the title of the page. In addition all events associated with an interface are identified in the structural object model. Some widgets such as the information widget (see line 20 in Script 1) have an implicit event associated with it which is picked up in the structural object model.

5.2.3 Deriving the Data Model

A data model can be derived from the structural object model. Figure 3 represents the data model derived from the structural object model in Figure 2. All data bearing widgets, except computationally dependent widgets, and widgets with "information" property or "confirmation" property are included in the data model. E.g. a widget such as "Accept usage terms" is not meant for supplying or accessing domain data. Domain tables are identified from container widgets. Thus two tables "Employees" and "Experiences" are identified. The relationship between tables in the data model is captured from the nesting of container widgets and the multiplicities are derived from keywords such as "table" in line 27 in Script 1.

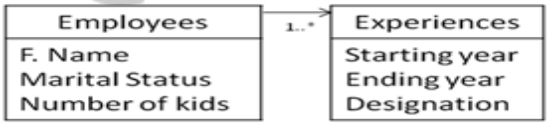

Figure 3: Initial data model of the application.

5.2.4 Language Support for the Behavioural Model

A behavioural model is required to support UI tasks. Script 2 contains a partial behavioural model of the application illustrated in Figure 1. The general form of the behaviour model is: "on a widget's event, apply reaction(s) over target widget(s)". End users may create the behavioural model by using the structural object model of the UI as the reference. The script uses UI widgets and events defined in the structural object model along with widget reactions. For example lines 6 to 8 in Script 2 specify that when the "set event" occurs on the "marital status data input widget", "property change reaction" is applied over "Number of kids data input widget". Similarly Script 2 may be continued for other event generating widgets.

Script 2: Partial behavioural model of the UI

```
01 In application <application_id>
02 on <employee details container page>
03 load event
04 apply data_request_reaction over
05 page <employee details>;
06 on<marital status data input widget>
07 set event
08 apply property change reaction over
09 <Number of kids data input widget>;
10 ...(continues for other widgets)
```

5.3 Language Support for Addition of Widgets

The changing business requirement can result in addition of new interfaces to an application. Figure 4 illustrates a case for evolution of an application by the addition of new widgets. The dotted horizontal line in the figure separates pre and post additions. The new additions are in the form of:

a) a "L. Name" text field for data viewing;
b) a navigation link from the "L. Name" label to the experiences page;
c) a new column, "Role" in the Experiences table.

The corresponding end-user model for the evolution of the application is presented in Script 3. The language details of Script 3 are not explained here as they are similar to that of Script 1.

The addition of new widgets causes consequential changes to the structural object model the details of which have not been provided for economy of space.

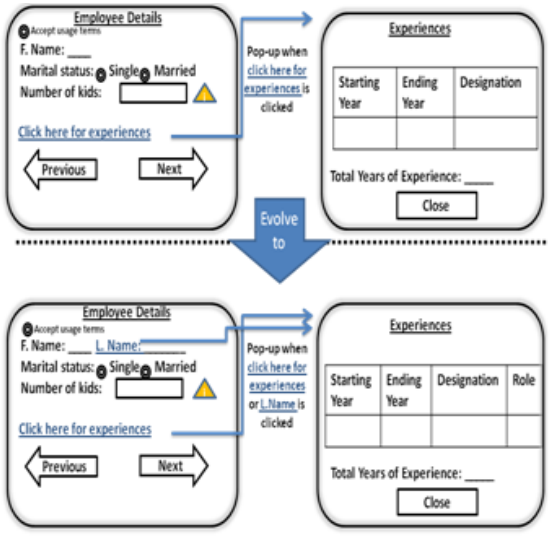

Figure 4: Evolution by addition of UI widgets.

Script 3: Addition of widgets

```
01 In application <application_id>
02 on page <Employee Details>(
03 for each <Employee> {
04  add <L. Name>
05  (string, read);
06  add <L. Name> (link , on click,
07  to pop-up page <experiences>),
08  after
09  <First Name>
10 });
11 on page <Experiences>(
12  add table( <Experiences>:<Role>
13  (string, read,
14  after
15  <Experiences>:<Designation>
16  ))
17 )
```

5.3.1 Effect of Addition on the Data Model

When new widgets are added, its effect on the data model must be captured. The data model gets affected only if new domain data bearing widgets are added to the UIs. Furthermore this may also result in the creation of new data tables if the user adds container widgets.

Script 3 causes changes to the initial data model in the form of alternations to the Employees and Experiences table structures. The altered data model is shown in Figure 5.

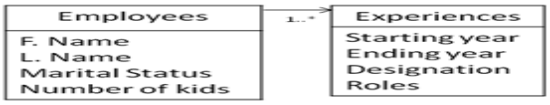

Figure 5: The altered data model.

5.3.2 Effect of Addition on the Behavioural Model

The behavioural model of the UI also gets affected due to the addition of new widgets. Users may recreate the behavioural model using the existing behaviour model as a template. New creation is preferred over editing of the existing behavioural model because additions of new widgets may affect the behaviour of old widgets too.

5.4 Language Support for Deletion of Widgets

The deletion of widgets involves deleting at least one widget from the UI. It does not mean deleting events or editing existing properties of a widget. Permissions to delete depend on the structural relationship types which are maintained in the

structural object model.

Widgets with no relationships can be safely deleted without affecting other widgets. Container widgets cannot be deleted unless its widgets are first deleted. The widgets in a container can be deleted if they are not involved in other relationships. Widgets in logical or computational relationships cannot be deleted unless their dependencies are taken care before the deletion. Similarly widgets in event relationships or pop-up relationships may be deleted only after the dependencies are removed. In summary, no widget can be deleted unless it has no relationships with other widgets.

5.4.1 Effect of Deletion on the Data Model

When a widget gets deleted, the structural object model is re-created. This may have an impact on the data model. E.g. if the deleted widget is a source widget for domain data, the corresponding data model field too will be deleted. Hence data-bearing widget deletion will have an impact on the domain data model, resulting in a logical deletion of a database table field. If a container widget is deleted it may result in the deletion of a database table itself. However physical deletions should be avoided to maintain the integrity of legacy data.

5.4.2 Effect of Deletion on the Behavioural Model

UI widget deletions will cause changes to the behaviour of the application. Hence the behavioural model of the system will have to be recreated. Further details are not provided since the modelling of the behavioural process is similar to that discussed earlier.

5.5 Language Support for Editing of Widgets

Editing of a widget means changing the properties of a widget or changing the events and the reaction to the events associated with an event generating widget.

Widgets with no relationships can be safely edited without affecting other widgets. Containers cannot be edited to be non container widgets unless it's containing widgets are first deleted. Widgets in computational relationships must be edited together. Similarly widgets in logical relationships must be edited together as the widgets are related by a logical expression.

Editing of an event generating widget involves either adding an event or changing an event. Changing an event can be perceived as deletion of an event followed by an addition of a new event. If an event is deleted from a navigation widget it may result in the navigational widget's target widget to be unattached to any other widgets of the page. However the target widget must not be inaccessible since it is not deleted. Such unlinked widgets must be maintained in the structural object model for possible future editing by the end-user.

5.5.1 Effect of Editing on the Data Model

The editing of widgets does not change the structure of the data model as no new widgets are added nor any widget deleted. Any change in the data model as a result of renaming of the widgets may be avoided by representing the database tables' names independently of the user-defined names of the associated interfaces. However the behavioural model needs redesign as in other cases.

5.5.2 Effect of Editing on the Behavioural Model

UI editing will cause changes to the behaviour of the application. However further details are not provided since the modelling of the behavioural process is similar to that discussed earlier.

6 CONCLUSIONS

This paper presents an end-user friendly UI modelling language for creating and supporting the evolution of RIA with changing requirements. The textual model of the UIs can be processed by an engine to derive the UI model for each page along with the underlying data model. For this we have identified various types of structural relationships among UI widgets which are important for managing evolution. When requirements evolve adding, editing or deleting widgets in a UI must be performed while managing existing relationships. The language described in the paper is end- user friendly. Further it is possible to derive the underlying data model from the UI specification. Future work will address engineering approaches and the evaluation of the efficacy of the language with respect to user-friendliness, generation of the structural relationships and end user understanding of the created models.

REFERENCES

Bozzon, A, Comai, S, Fraternal, P & Carughi, GT. 2006. Conceptual Modeling and Code Generation for Rich Internet Applications. *International Conference on Web Engineering*, July 11 - 14 2006 Menlo Park, California, USA. ACM, 353-360.

Busch, M & Koch, N 2009. Rich Internet Applications: State-of-the-Art. Munich, Germany: Institute for Informatics, Ludwig-Maximilians-Universität.

Calvary, G, Coutaz, J, Thevenin, D, Limbourg, Q, Bouillon, L & Vanderdonckt, J 2003. A Unifying Reference Framework for multi-target user interfaces. *Interacting with Computers,* 15**,** 289-308.

Cypher, A 1993. Watch What I Do: Programming by Demonstration. Cambridge, Massachusetts & London, England: MIT Press.

Farooq Ali, M, Pérez-quiñones, M A & Abrams, M 2005. Building Multi-Platform User Interfaces with UIML. *Multiple User Interfaces.* John Wiley & Sons, Ltd.

Garrigós, I, Meliá, S & Casteleyn, S 2009. Adapting the Presentation Layer in Rich Internet Applications. *Web Engineering*. Springer Berlin / Heidelberg.

Griffiths, T, Barclay, P J, Paton, N W, McKirdy, J, Kennedy, J, Gray, P D, Cooper, R, Goble, CA & da Silva, PP 2001. Teallach: a model-based user interface development environment for object databases. *Interacting with Computers,* 14**,** 31-68.

JavaFx. 2007. *JavaFx* [Online]. Oracle. Available: http:// javafx.com/ [Accessed March 19 2012].

Ko, A J, Abraham, R, Beckwith, L, Blackwell, A, Burnett, M, Erwig, M, Scaffidi, C, Lawrance, J, Lieberman, H, Myers, B, Rosson, MB, Rothermel, G, Shaw, M & Wiedenbeck, S 2011. The state of the art in end-user software engineering. *ACM Comput. Surv.*, 43**,** 1-44.

Liang, X D & Ginige, A. 2007. Enabiling an End-User Driven Approach for Managing Evolving User Interfaces in Business Web Applications: A Web Applicaiton Architecture using Smart Business Object. International Conference on Software and Data Technologies, 2007 Barcelona. 70-78.

Limbourg, Q, Vanderdonckt, J, Michotte, B, Bouillon, L & López-Jaquero, V 2005. USIXML: A Language Supporting Multi-path Development of User Interfaces Engineering Human Computer Interaction and Interactive Systems. *In:* Bastide, R, Palanque, P & Roth, J (eds.) *Engineering Human Computer Interaction and Interactive Systems.* Springer Berlin / Heidelberg.

Little, G & Miller, R C. 2006. Translating keyword commands into executable code. ACM Symposium on User Interface Software and Technology, 2006. 135–144.

Liu, H & Lieberman, H. 2005. Programmatic semantics for natural language interfaces. ACM Conference on Human Factors in Computing, 2005. 1597–1600.

Melia, S, Gomez, J, Perez, S & Diaz, O 2010. Architectural and Technological Variability in Rich Internet Applications. *Internet Computing, IEEE,* 14**,** 24-32.

Puerta, A & Eisenstein, J. 2003. Developing a Multiple User Interface Representation Framework for Industry. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.145.6974&rep=rep1&type=pdf [Accessed 28 April 2012].

Silverlight. 2010. *Get Started with Silverlight* [Online]. Microsoft. Available: http://www.silverlight.net/ [Accessed March 19 2012].

Urbieta, M, Rossi, G, Ginzburg, J & Schwabe, D. 2007. Designing the Interface of Rich Internet Applications. 5th Latin American Web Congress (LA-Web'07), 2007 UNLP, Buenos Aires. *IEEE*, 144-153.

Valverde, F, Panach, I, Aquino, N, Pastor, O, Macías, JA, Granollers Saltiveri, A & Latorre, PM 2009. Dealing with Abstract Interaction Modeling in an MDE Development Process: A Pattern-Based Approach: *New Trends on Human–Computer Interaction*. Springer London.

Valverde, F & Pastor, O 2009. Facing the Technological Challenges of Web 2.0: A RIA Model-Driven Engineering Approach *Web Information Systems Engineering - WISE 2009.* Springer Berlin / Heidelberg.