Designing Rich Internet Applications Combining UWE and RUX-Method

Juan Carlos Preciado, Marino Linaje, Rober Morales-Chaparro, Fernando Sanchez-Figueroa
Quercus SEG. Universidad de Extremadura
{jcpreciado, mlinaje}@unex.es

Gefei Zhang[1], Christian Kroiß[1], Nora Koch[1,2]
Web Engineering Group
[1]*Ludwig-Maximilians-Universität München*
[2]*Cirquent GmbH*
{zhangg,kroiss,kochn}@pst.ifi.lmu.de

Abstract

The rapidly increasing importance of Rich Internet Applications (RIAs) calls for systematic RIA development methods. However, most current Web engineering methods focus on Web 1.0 applications only; RIAs, on the contrary, are still developed in an ad-hoc manner, which often results in error-prone and hard-to-maintain products. We propose a model-driven approach to RIA development by combining the UML-based Web Engineering (UWE) method for data and business logic modeling with the RUX-Method for the user interface modeling of RIAs.

1. Introduction

One of the most exciting recent movements of Web applications is the trend towards Rich Internet Applications (RIAs). RIAs introduce features and functionality of traditional desktop applications like animations and client-side computing to Web applications. The advantages of such Web applications include complex user interactions and the overcoming of page-loading requirements of traditional Web 1.0 applications. Enterprises are rapidly adopting Web 2.0 features like RIAs as they see high business value in the innovation [2] [9]. However, there is still a lack of engineering methods for RIAs [11]. Most current Web engineering methods consider only Web-1.0 features. Consequently, RIAs still have to be developed in an ad-hoc manner. The inadequateness of abstraction and documentation makes them error-prone and hard–to-maintain. We propose to address this problem by combining the UML-based Web Engineering approach (UWE) [4] [5] with the RUX-Method [7] for the development of RIAs.

UWE provides a domain specific notation for the graphical representation of Web applications and a method for the model-driven development of Web systems. The RUX-Method is a model-driven approach to modeling the User Interface (UI) of RIAs. It may be used on top of any Web engineering method. The RUX-Method replaces the original presentation model by a new RIA one. In our approach, UWE is used to specify the content, navigation and business processes of the Web application, and the RUX-Method is used on top of these models to add typical rich UI capabilities, such as temporal behavior and rich user interactions. We build the bridge between both approaches defining transformation rules between their meta-models.

While the RUX-Method has already been combined with WebML [12], the novel idea presented in this paper is the extension of the generation rules of the underlying method (UWE) in order to obtain the connection with the RUX-Method automatically. The connection provides the mechanisms needed to generate the UI step by step. Furthermore, our approach is to the authors' knowledge the first one for the development of RIAs including business processes.

The rest of this paper is structured as follows: Section 2 summarizes the background of the proposal using a running example. Section 3 presents the connection between UWE and the RUX-Method. Section 4 shows related work, and finally, we outline some future work in Section 5.

2. Background

In this section we present briefly UWE and the RUX-Method by an example. Consider a simple Web database of movies, which provides the information of a collection of movies (see Figure 1). The user can browse to the detailed information about a movie through an index, add a new movie to the database, or remove a selected movie from the database.

2.1. UWE in Brief

UML-based Web Engineering (UWE [4]) is a method for systematic and model-driven development of Web applications. UWE follows the principle of "separation of concerns" by modeling the content, the navigation structure, the business processes, and the

presentation of a Web application separately. UWE implements a model-driven development process by defining model transformations of different types to derive platform specific models from platform independent models and to generate running programs [6].

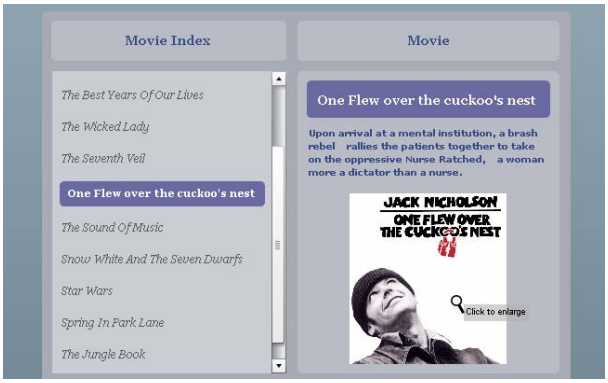

Figure 1. Example: online movie database

UWE's outstanding feature is its reliance on standards: its modeling language is defined by an extension of the Unified Modeling Language metamodel (UML 2.0 [10]) and mapped to a so-called UML profile; its transformations are defined in (on coming) standard transformation languages like QVT [10] or ATL [1].

The UWE design process starts with a requirements model that comprises use cases, discerning navigational from business process use cases. The requirements model of our example includes a «navigation» use case for browsing the movie database and standard use cases for adding and removing movies. The content model in UWE – represented by a normal UML class diagram – provides a specification of the domain-relevant information for the Web software. The content model of the movie database contains movies that are organized in movie collections. Movie collections are characterized by some genres. Each movie has a title, a description, some photo, a genre, and a release year. The classes MovieCollection and Movie include methods for adding a movie to or removing a movie from a movie collection.

Based on the requirements and the content model, the navigation model of the Web application is built to specify the hypertext structure of the system, which is given by nodes and links. Classes with stereotype «navigation class» (like MovieCollection or Movie in Figure 2) represent navigable nodes for information retrieval; «process class»es (like AddMovie and RemoveMovie) define navigation nodes where transactions may occur. Direct links are modeled by associations; in particular, «process link» stereotyped associations lead to or leave from process classes.

Some special navigation nodes are used to organize links. For example, several instances of a navigation class are reached by an «index» (like MovieIndex) and choices of links are represented by «menu»s (like MovieCollectionMenu). In our movie database, users can navigate from the movie collection via an index of the movies to view the information of a selected movie or, along another navigation path, add or remove some movie to or from the database (Figure 2).

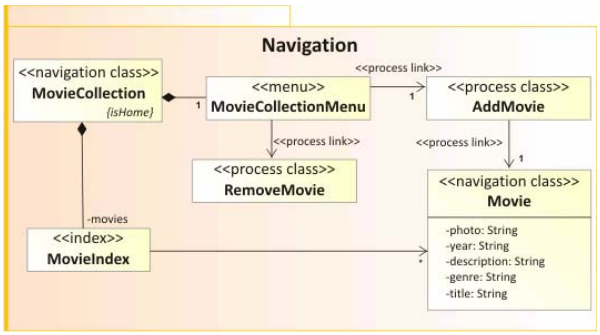

Figure 2. UWE: navigation model of the example

Each process class in the navigation model is refined by a process structure model in the form of a class diagram, defining additional classes used in the process, and a process flow model in the form of a UML activity diagram, modeling the data and control flow of the process. In particular, «user action» stereotyped actions indicate input from the user of the Web application.

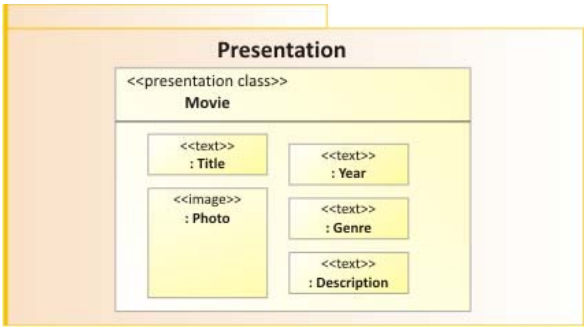

Figure 3. UWE: presentation model (excerpt)

The presentation model provides an abstract view of the Web-1.0 user interface (UI), where concrete aspects of the UI, such as colors and fonts of UI elements are not considered. For each navigation class, a «presentation class» models its presentation. UI elements, such as «text»s, «image»s etc., contained in presentation classes indicate the abstract type of the widgets to use.

Presentation classes can be nested, modeling the hierarchical structural of Web pages. A presentation class that is not contained in another represents a top-level page of the Web application. In addition, a presentation class is defined for each user action. The pre-

sentation model therefore has the form of a forest of presentation classes. An excerpt of the presentation model for the movie database is given in Figure 3.

2.2. RUX-Method in Brief

The RUX-Method (Figure 4) is a model-driven method for the systematic specification of multi-device and interactive multimedia Web UIs. The method can be combined with other Web methods which model the data and business logic of the Web application.

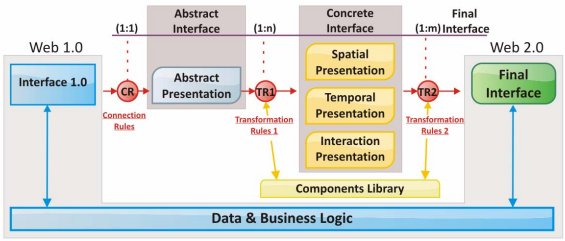

Figure 4. Overview on the RUX-Method

The RUX-Method distinguishes three different interface levels providing a conceptual chain of refinement [3][7]: abstract interface, concrete interface and final interface. Abstract interface provides a UI representation common to all RIA devices and development platforms, without any kind of spatial, look and feel or behavior dependencies. Since this interface is the most important in the context of this work more details will be given in Section 3.2. The concrete interface is platform independent but specific for a device or group of devices. It is divided into three presentation levels: spatial, temporal and interaction presentation.

Since the abstract interface provides a first draft of components grouping, in the spatial presentation the modeler simply needs to refine this grouping, specify the spatial arrangement of components, and define their dimensions and look and feel. The temporal presentation allows the specification of behavior which requires a temporal synchronization (e.g. animations). The interaction presentation allows the specification of the user's behavior with the RIA UI. In RIAs, capturing the user interaction with the UI is generally carried out by the application components that are able to capture certain event types.

The final interface contains the information for the UI code generation which is specific for a device or a group of devices and for a RIA development platform such as FLEX, Ajax or Laszlo.

In accordance to the three interface levels, there are also three transformation phases in the RUX- Method. The first transformation phase catches and adapts the data and business logic specified in the underlying (Web-1.0) models to the RUX-Method abstract interface, and is called *connection rules* (marked as **CR** in Figure 4). The abstract interface is adapted in the second transformation phase to one or more particular devices and grants access to the business logic. This phase is called *transformation rules 1* (marked as **TR1** in the Figure). Finally, in *transformation rules 2* (marked as **TR2** in the Figure) the MDA life-cycle of the RUX-Method is completed by code generation.

These transformations are based on a component library (see Figure 4). Each RUX-Method interface level is composed by interface components whose specifications are stored in the library. The library also allows the RUX-Method to keep the definition of several target platforms in a single XML document, as well as the translation of an origin component to one or more target platforms. For example, for the final interface, each component in the library is defined for different rich rendering platforms.

Once the abstract interface is obtained by applying the CRs to the underlying Web model, further refinements towards concrete interface and final interface can be made, so that finally a RIA implementing the same functionality but providing a much better user friendly UI can be generated.

3. Connection of UWE and RUX-Method

The model-driven nature of UWE and RUX-Method makes it straight-forward to extend the generation rules of UWE to obtain the CRs' specification automatically; separating presentation from other aspects of Web applications like content and navigation structure allows us to localize these extensions, instead of having to change the Web-1.0 UWE models ubiquitously.

3.1. Metamodel-based Generation of Web Applications in UWE

UWE's generation process is metamodel-based: it considers UWE models as instances of the UWE metamodel, which is an extension of the UML metamodel, and maps them to instances of platform-specific metamodels, from which running code is generated. In particular, the classes in the content model are implemented either as Java Beans or by RMI; the presentation classes are transformed to Java Server Pages (JSPs); information contained in the navigation model is transformed to configuration files of the generated application. Process models are not transformed, but directly executed by means of the Spring framework[1] by the runtime environment.

[1] www.springframework.org

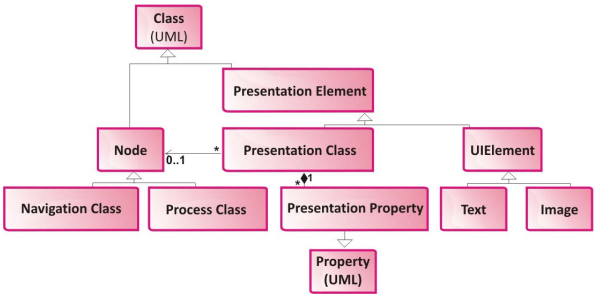

Figure 5. UWE metamodel: presentation package (excerpt)

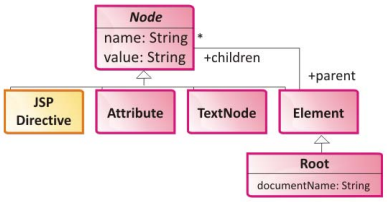

Figure 6. JSP metamodel [6]

For example, the rule below specifies the transformation of presentation models (metamodel given in Figure 5) to JSP models (metamodel in Figure 6): each "top-level" presentation class (that is, each presentation class that is not included in another) is mapped to a Root element (of a JSP document), implementing the outer structure of an HTML page; the presentation classes contained in the top-level presentation class are mapped recursively to UI elements contained in the body of the JSP.

```
rule PresentationClass2JSP {
  from
    pc : UWE!PresentationClass (pc.isTopLevelPC())
  to
    jsp : JSP!Root(documentName <- pc.name + '.jsp',
      children <- Sequence{ htmlNode }),
    htmlNode : JSP!Element(name <- 'html',
      children <- Sequence{ headNode, bodyNode }),
    headNode : JSP!Element(name <- 'head',
      children <- Sequence{ titleNode }),
    titleNode : JSP!Element(name <- 'title',
      children <- Sequence{ titleTextNode }),
    titleTextNode : JSP!TextNode(value <- pc.name),
    bodyNode : JSP!Element(name <- 'body',
      children <- pc.ownedAttribute->collect(p | p.type))
}
```

The helper function isTopLevelPC() determines if a presentation class is a top-level one. Its specification is as follows:

```
helper context UWE!PresentationClass def :
isTopLevelPC() : Boolean =
  not UWE!PresentationClass.allInstances()->
        exists(pc | pc.ownedAttribute->
          exists(p | p.type = self));
```

3.2. RUX-Method Abstract Interface Design

The RUX-Method abstract interface is composed of three different kinds of elements: connectors, media and views.

- **Connectors** are used to establish the relation between the UI component and the data represented in the content model;
- **Media** elements represent device-independent atomic UI information. They are categorized into discrete media (texts and images) and continuous media (videos, audios and animations);
- **Views** are used to group the information that will be shown in the UI. The RUX-Method distinguishes four different types of containers: simple, alternative, replicate and hierarchical Views.

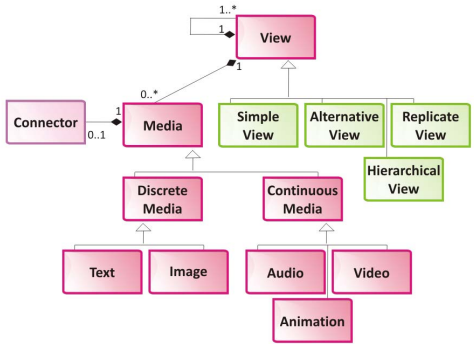

Figure 7. RUX-Method Abstract I. metamodel

In the RUX-Method, the root element of the abstract interface is always a View. When View type is Simple it may contain other Views and/or Media elements. Alternative, replicate and hierarchical views cannot contain Media elements.

An alternative View indicates that only one of the views that it contains will be shown at the same time to the user (e.g. Tab panels). The replicate View determines other Views that are going to be repeated throughout the View that contains it (e.g. list of elements). The hierarchical View represents elements in a tree view (e.g. a set of categories and sub-categories). The metamodel of the RUX-Method abstract interface is given in Figure 7.

3.3. Connecting RUX-Method with UWE

The CRs connecting UWE and the RUX-Method must filter the information presented in UWE and extract the information needed to build the abstract interface and to trigger the business logic rightly. As we will show, the UWE presentation model is enough for both issues.

From the UWE presentation model, the CRs extract the relationship among UI elements and their composition hierarchy, distinguishing between contents and containers, identifying the Media elements and grouping the containers using the options according to the RUX abstract interface metamodel.

In the following, the information extraction and propagation from UWE to the RUX-Method is described in more detail. We also show how this process can be automated by extending the ATL generation rules of UWE.

I. Extracting the Hierarchy of Composition from UWE Models

This process is described below using pseudo-code, organized in two steps. The first step creates an empty RUX-Method abstract interface (marked as ⓐ in Figure 8). In the second step, each UWE «presentation class» and the UI elements that it contains are then added recursively as children to this SimpleView (ⓒ and ⓓ).

Start
Create a SimpleView as the root element.
For each p, p is of type «presentation class», «text» or «image» contained in the presentation model
    Create a **SimpleView V** inside the last view we have created from the **p.parent,** or inside the **root** if p has no parents.
    If **p** is a «presentation» node:
        Create a **Connector** that references to **p** inside **V**
        If **p** has relationships with a multiplicity of..* (unbounded) at the children role
            Create a **Replicate View R** inside **V,** in order to use the **R** as the parent for the results of applying CR to **p**'s children
    ElseIf **p** is a «text», «image», «anchor», etc. node:
        Create a **Media** inside **V**.
        Connect the **Media** with the **Connector** created from **p.parent**.
        Select the type of **Media** conveniently (direct since UWE and the RUX-Method support the same types of media).
        Specifically indicate which attribute of the **Connector** use the **Media**. This can not be inferred so it is required because there is only *one* Connector for every UWE <<presentation>> node, and it has usually more than one attribute.
    EndIf
End

II. Collecting all Operation Links or Operation Chains

Once the abstract interface has been specified and TR1 have been applied, the RUX-Method concrete interface allows modeling according to the device specific capabilities.

As depicted in Figure 8 (marked as ⓑ), the CRs also retrieve and propagate to the concrete interface the information required to build a List of Useful Links (LUL) according to the different kinds of actions defined in the RUX model, (e.g. UIActions or CallActions) [7]. Since UWE models Web 1.0 applications, the elements contained in the LUL of UWE are all CallActions, which model simple user interaction like mouse clicks to follow a hypertext link. So LUL consists of the «navigation link»s and «process link»s in the UWE navigation model, as well as the links in the process flow model leading to and leaving from «user action»s.

When the RUX-Method applies the TR1 to get the concrete interface from abstract interface, we need to take into account the information offered by UWE regarding operation chains. The algorithm is as follows:

For each LINK ("L") listed in **LUL**
    Create a handler with name "L"
    Add a CallAction to reference "L"
For each Media "M" in the abstract interface:
    Add a listener for every output "M" (RUX-Method Media elements can be for input purposes e.g. combobox or for output e.g. label), called **O**
    For each listener **O**
        Set the handler descriptor to be the same as created before.
        Connect with the first event defined in the component library (default event) for the component (usually, **click**)

III. ATL Rules of the Connection

Since the transformations in UWE (see Sect. 3.1) are first-class citizens and therefore can be modified or extended to meet new requirements, it is straightforward to include the connection with the RUX-Method designs as described above into the UWE generation process. Generating the RUX-Method abstract interface model from UWE presentation model automatically reduces the cost of the connection greatly by avoiding many easily made errors when doing it manually. For instance, the following (simplified) transformation generates for each top-level presentation class in UWE a simple view in the RUX-Method.

```
rule PresentationClass2SimpleView {
 from
      pc : UWE!PresentationClass (pc.isTopLevelPC())
 to
      sv : RUX!SimpleView(
              children <- pc.ownedAttribute->select(p |
                 p.type.oclIsKindOf(UWE!PresentationClass
                    )),
                media <- pc.ownedAttribute->select(p |
                      p.type.oclIsKindOf(UWE!UIElement))
}
```

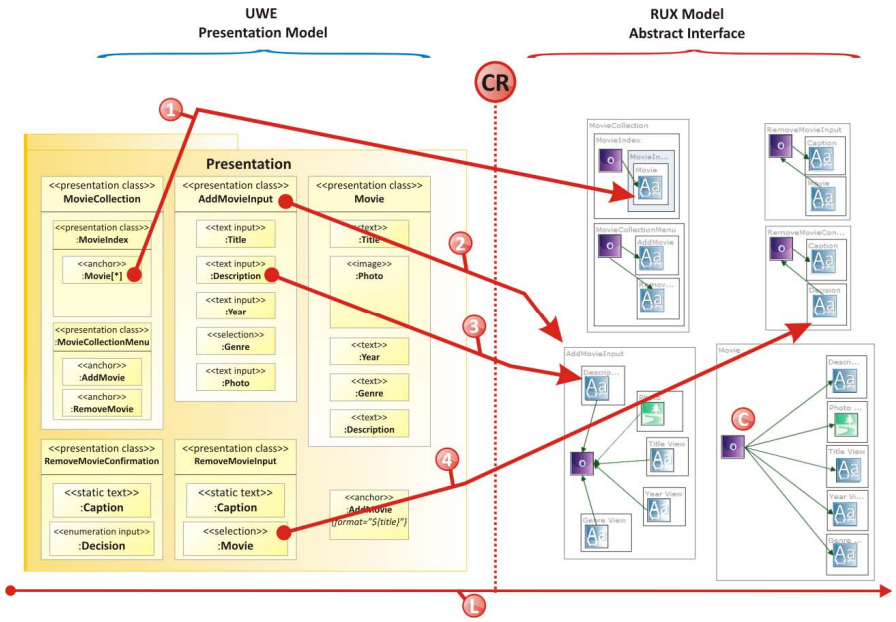

Figure 8. The connection process for the running

3.4. The Running Example

Figure 8 depicts some of the design phases for our running example. Note that CRs contain information from UWE presentation models to design a rich user interface for it.

In the center of Figure 8, the relationship between UWE models and the RUX-Method abstract interface is depicted, showing these CRs graphically (marked as ❶, ❷, ❸, and ❹). This figure also depicts the propagation to extract the list of links to trigger the business logic (marked as Ⓛ). The resulting abstract interface of the example built with the RUX-Method is shown on the right side where different types of Views and Media are depicted, as well as Connectors (e.g. the one marked as Ⓒ).

For our movie database example, Figure 8 shows how the CRs have placed an alternative root view containing a SimpleView for each presentation class of UWE presentation root level. In this example, these SimpleViews are related with MovieCollection, AddMovieInput, RemoveMovieInput and Movie, using the same names which have been automatically obtained (e.g., ❷). Many of the abstract interface groupings have also a Connector in which the relationship between the UWE presentation model and the RUX-Method is specified.

Afterwards, for every one of these presentation classes, the content placed in this kind of containers will be processed using the extracted UWE presentation elements through the connection process (e.g., ❷, ❸ and ❹). In the Figure, the arrow ❸ shows how the «text input» Description of the presentation class AddMovieInput is represented by means of a text Media inside the Description SimpleView (and so on for the rest of the Media as the one marked as ❹). Finally, also the relations between Media and Connectors are available to store the specification of which attribute of the connector each media is connected to. Using the direction of the arrows, this relationship also provides the specification if it is an output (the relation is from the Media to the Connector –e.g. in the AddMovieInput SimpleView) or an input Media (Connector to Media – e.g. in the Movie SimpleView).

4. Related Work

In [11] the problems of the different methodologies from Web and Multimedia fields when considering RIAs are shown. The work in [8] proposes a first draft of a model driven method for designing graphical user interfaces in RIAs decomposing the presentation design into several abstraction levels. The method is based on XSL model transformations.

Toffetti et al. propose in [13] the modeling of distributed events in data-intensive RIAs. They show how events can be explicitly described and coupled to the other concepts of a Web modeling language in order to specify collaborative RIAs. They apply these concepts to WebML. Issues related to behavior, single-page paradigm and content composition are treated in [14]. This work conceptually extends the method OOHDM for modeling RIAs supporting very abstract spe-

cification of RIA user interface. Platform specific generation is not considered.

Our proposal supports the development of business process driven RIAs. We demonstrated that, by extending the generation rules of UWE, it is possible to obtain automatically the connection rules of the RUX-Method, which makes it an easy task to adapt the user interfaces of Web 1.0 applications modeled in UWE to multi-device RIA UIs.

5. Conclusions and Future Work

We have presented a model-driven approach to RIA development by connecting UWE and the RUX-Method. UWE is more appropriate for modeling the functionalities, i.e. data and business logic, of Web applications; the RUX-Method is applicable for designing the RIA user interface. Our approach provides a simple way to enrich Web 1.0 applications with Web 2.0 look and feel. Connection rules are formulated in the ATL language and thus the connection can be established, i.e., a RUX abstract interface can be created, automatically. Our approach considers both static navigation and dynamic business process.

Our future work includes enhancing this approach by including RIA specific operations, such as automatic completion of input fields or client side computing, that are not possible in Web 1.0 and thus not considered by the current Web (1.0) engineering methods. We plan to extend the RUX-Method by model elements, which, e.g., represent the client sending requests to the server in the background or carrying out some operation, and to extend UWE by model elements which represent the corresponding server operations. The planned extensions in both methods as well as their connection are expected to be straightforward.

Another important issue in RIA engineering is requirements engineering. We plan to extend the UWE requirements description techniques to include RIA features like animation, asynchronous client server communication, client side computing, etc.

Acknowledgments.

This research has been partially supported by the following projects: MAEWA (WI841/7-1) of the DFG, Germany, the EC 6th Framework project SENSORIA (IST 016004), and the Spanish Government project TIN2005-09405-C02-02.

6. References

1. ATLAS Transformation Language and Tool, http://www.eclipse.org/m2m/atl/doc/
2. Bozzon, A., Comai, S., Fraternali, P., Toffetti Carughi, G.: Conceptual Modeling and Code Generation for Rich Internet Applications. In Proc. of the 6th Int. Conf. on Web Engineering, pp. 353-360, ACM (2006)
3. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A Unifying Reference Framework for Multi-Target User Interfaces. Interacting with Computers, vol. 15, no. 3, pp. 289-308 (2003)
4. Koch, N., Knapp, A., Zhang, G., Baumeister, H.: UML-Based Web Engineering: An Approach Based on Standards. In Web Engineering: Modelling and Implementing Web Applications, HCI Series, vol. 12, chapter 7, pp 157-191, Springer-Verlag (2007)
5. Koch, N., Kraus, A., Cachero, C., Meliá, S.: Integration of Business Processes in Web Application Models. Journal of Web Engineering, vol.3 pp. 22-49 (2004)
6. Kraus, A.: Model Driven Software Engineering for Web Applications. PhD. Thesis. LMU München (2007)
7. Linaje, M.; Preciado, J.C.; Sánchez-Figueroa, F.: Engineering Rich Internet Application User Interfaces over Legacy Web Models. Internet Computing Magazine, IEEE, vol.11, no.6, pp.53-59 (2007)
8. Martínez-Ruiz, F.J., Muñoz Arteaga, J., Vanderdonckt, J., González-Calleros, J.M.: A First Draft of a Model-driven Method for Designing Graphical User Interfaces of Rich Internet Applications, In Proc. of the 4th Latin American Web Congress, pp. 32-38, IEEE (2006)
9. Murugesan, S.: Understanding Web 2.0. IT Professional Journal, vol.9, no.4, pp.34-41 (2007)
10. OMG, http://www.omg.org
11. Preciado, J.C., Linaje, M., Sánchez-Figueroa, F., Comai, S.: Necessity of Methodologies to Model Rich Internet Applications, In Proc. of the 7th IEEE Int. Symp. on Web Site Evolution, pp. 7-13, IEEE (2005)
12. Preciado, J.C., Linaje, M., Sánchez-Figueroa, F.: An Approach to Support the Web User Interfaces Evolution. In Proc. of the 2nd Int. Workshop on Adaptation and Evolution in Web Systems Engineering, ICWE, pp. 94-100 (2007)
13. Toffetti, Carughi G., Comai, S., Bozzon A., Fraternali, P.: Modeling Distributed Events in Data-Intensive Rich Internet Applications, In Proc. of the 7th Int. Conf. on Web Information Systems Engineering, LNCS 4607, pp. 593-602, Springer-Verlag (2007)
14. Urbieta, M., Rossi, G., Ginzburg, J., Schwabe, D.: Designing the Interface of Rich Internet Applications. In Proc. of the 5th Latin American Web Congress, pp.144-153, IEEE (2007)