# Patterns for the Model-Based Development of RIAs*

Nora Koch[1,2], Matthias Pigerl[3], Gefei Zhang[1], and Tatiana Morozova[1]

[1] Ludwig-Maximilians-Universität München, Germany
[2] Cirquent GmbH, Germany
[3] S.CO LifeScience GmbH, Germany

**Abstract.** Rich Internet Applications (RIAs) are highly interactive web applications that resemble desktop applications. Modeling RIAs hence requires techniques for web modeling enriched by model elements for powerful user interactions and client-server communications. Many existing approaches provide the required modeling features, but they are still failing short in designer-friendliness and effectiveness. We present a pattern approach for the model-based engineering of RIAs that (1) reduces design efforts maintaining expressiveness of the models, and (2) contributes to model-driven development of RIAs. Our RIA patterns can be easily embedded in existing web modeling methods, which is illustrated with the UML-based Web Engineering.

## 1 Introduction

Rich Internet Applications (RIAs) are web applications augmented with desktop features and mechanisms of advanced communications. The rich look and feel, better responsiveness, performance, and accessibility enthuse both users and software providers. RIAs improve user interaction facilities like drag&drop, multimedia presentations and avoid unnecessary page reloading. Data handling and operations executed on the client side minimize server requests.

RIAs use the power of client and server and can be implemented using different techniques, such as Asynchronous Javascript with XML (AJAX) for which a set of frameworks have been developed (Flex, Ruby on Rails, etc.). Frameworks are quite helpful for programming, but the development of software requires also support for other phases like design and testing. In particular, model-driven development requires building models during the design and using models as sources of transformations to generate other models or running code. In this paper we focus on modeling RIAs.

Different modeling approaches for RIAs were developed as extensions of existing methods during the last few years (see Sect. 5). The extensions proposed so far consist of creating new model elements for the new RIA features, such as the validation of an input field value as soon as the user moves the mouse out of it. However, they do not alleviate the designer's work, i.e. provide the appropriate model elements of frequent use in the domain with precisely defined semantics.

* This research has been partially supported by the projects MAEWA II (WI841/7-2) of the DFG, Germany, and the EC 6th Framework project SENSORIA (IST 016004).

M. Gaedke, M. Grossniklaus, and O. Díaz (Eds.): ICWE 2009, LNCS 5648, pp. 283 – 291, 2009.

Patterns have proved valuable for efficient RIA programming [7]. We propose to apply patterns at a higher abstraction level, i.e. modeling, to achieve the objectives of minimizing the design efforts and maximizing the expressiveness of the models used in the development of RIAs. Our focus is on the use of state machines for the representation of the patterns – a widely used modeling technique. The models of the RIA patterns we specify can be embedded in almost all existing methodologies. In this sense, it is a general approach for all UML conform methods. The use of these RIA patterns only requires the definition of extension points in the methodology, and afterwards the specification of how to integrate the patterns, which makes our patterns easily reusable. In this paper and for demonstration purposes only, we use the UML-based Web Engineering (UWE) [4] as the hosting language.

This paper is organized as follows: Section 2 describes the industrial case study S.CORE used to validate the approach. Section 3 presents RIA patterns. Section 4 describes the integration of RIA patterns in the development using existing web modeling methods; the procedure is illustrated by UWE. In Section 5 we discuss some related work. Section 6 concludes and provides some ideas for future work.

## 2 Real World Case Study: S.CORE System

Our approach was validated in a real case study: the S.CORE application of the company S.CO LifeScience [11]. S.CORE is a web-based image analysis system. Since the complete service of image analysis is offered totally through the web, no software needs to be downloaded. S.CORE offers standard analysis modules such as cell counting, as well as customized solutions tailored to the customers' need for enumerating, measuring and statistical quantification of images. S.CORE is mainly used in the lifescience area.

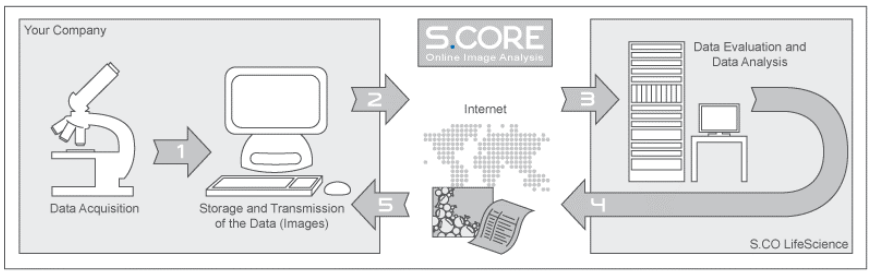


**Fig. 1.** Image analysis by S.CORE system

The image analysis process supported by S.CORE is shown in Fig. 1. It consists of the following steps: (1) The customer obtains and prepares the digital images of his samples with the appropriate technical tools in his laboratory. (2) With the help of S.CORE, which has a web-based human-machine interface, configured to the specific requirements of the customer, he can upload the images and specifications for performing the desired analysis type in the processing center of S.CO LifeScience. (3) The images are loaded automatically into the Analyzer of S.CORE, where the desired

data is extracted from the image and made available on the server. (4+5) The customer can download the results (including the processed image).

Within a collaboration between the Ludwig-Maximilians-Universität München and S.CO LifeScience, the S.CORE system was augmented with Web 2.0 features. This improved the user-friendliness of S.CORE. In particular, input fields supported by auto-completion, the progress of the upload and download functions visualized, images to be analyzed presented as a gallery of images, the result page should show the current status of analysis results. These Web 2.0 features challenged the design of the system with UWE.

## 3 RIA Patterns

Best practices for the implementation of RIAs include the use of RIA patterns. A RIA pattern is a general reusable solution to a commonly occurring problem in RIA design. It describes the interaction, operation and presentation of a RIA widget [12]. The interaction is the trigger of the RIA pattern, i.e. every pattern starts with a user event or a system event, e.g. *mouseover*, *onfocus*, *keyboard stroke* or *time event*. A variety of operations can be triggered by the interaction, such as validate, search and refresh. Finally, the result of the operation implies an update of the user interface.

Bill Scott emphasizes that patterns are great for forming a design vocabulary [12]. However, we stress that these patterns if appropriately generalized and modeled, are more powerful as they could be integrated in the models already built for other concerns of a web application, e.g. presentation and process. This means that the models of the RIA patterns are solutions that could be reused in the development of RIAs. Reuse can be performed manually or in an automated development process, such as model-driven development by model-to-model transformations.

Our pattern-based approach for implementing RIAs then consists in the use of the models of RIA widgets and the embedding of these models of RIAS into the other existing models. Therefore the designer needs (1) models of the patterns describing the RIA features, (2) an event language for the representation of user and system events, (3) model elements to be integrated in the existing web application models, such as the presentation model to indicate which pattern should be applied in which case. Our pattern approach compared to other RIA modeling techniques concern only the abstract presentation. Concrete presentation issues, like multimedia, animation, etc., are handled in the concrete presentation model, for further details see [5]. The separation of abstract and concrete presentation is essential to an MDD process since the abstract presentation model can be easily reused for other platforms [10].

An event language is defined, which includes constructs such as mouseover, onblur, onfocus, onclick, etc. A RIA pattern catalog is created which contains RIA patterns for modern web applications used in S.CORE reference projects [9]. Note this catalog is not exhaustive. It is an open document, which will grow up in the way RIA patterns become more familiar. We illustrate our approach presenting in the following two patterns of the catalog [9].

**Auto-Completion.** A S.CORE user has to deal with a lot of forms and input fields, There are forms for analysis parameter, search masks or user registration. Similarly to forms of other web applications, some input fields ask for redundant data or data

which could be identified after entering the first characters. Unnecessary work would be burdened to the user if he had to fill in every input field. Therefore the application should fill in redundant input fields automatically. Here some examples to concretize the scenario: When a user starts entering the image name, the system should find the analysis name automatically if this image only exists in one analysis (Fig. 3). Obviously all data captured by application logics could only be suggestions in order to allow a better usability. The user is still the last instance for checking the correctness of the suggested data. Therefore he should be able to overwrite every auto-completed input field.

*Problem.* How could the user of a RIA get an immediate suggestion for values in an input field, after he has entered some initial data or has filled in other related fields?

*Motivation.* Assistance by filling in input fields and forms. (1) People make mistakes when they type. (2) Typing in data is a tedious work. (3) Typing speed remains a bottleneck; faster user input is aimed by reducing the number of keystrokes.

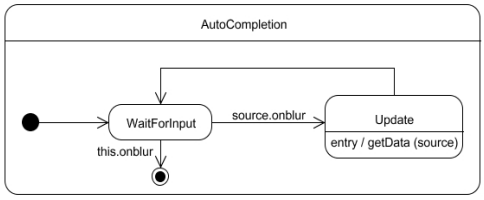


**Fig. 2.** RIA Pattern auto-completion (UML state diagram)

*Solution.* Suggest words or phrases that are likely to complete what the user is typing. As soon as the user moves to another input element or even as soon as the user inputs a character, the RIA in the background will try to query databases and find relations to already entered data. If such data could be found and the user has not yet completed it himself, a completed value for the input field is suggested to user. Fig. 2 shows the general pattern for auto-completion: on *source* (e.g. analysis name) losing focus (*onblur*), the RIA goes into the state *Update* to get the relevant data (image name), and then to update the corresponding content of itself. Note that the user can overwrite the suggested value as the widget remains in the *WaitForInput* state.

*Examples.* (1) Online banking. (2) Sixt car rental system (www.sixt.com).

**Periodic/Dynamic Refresh.** An S.CORE user can upload several microscope images for image analyses in order to extract statistical data from the uploaded images. Thus inside S.CORE there is a status page where the user can monitor the status and check the approximate finishing time of every analysis process. During the analysis the process runs through different states and also the approximation of the finishing time could change due to unpredictable analysis process events or other analysis with a higher priority being started. The status page should always provide correct and up-to-date information without a permanent reload of the whole page, i.e., change the list of analysis that are currently running, the remaining time for finishing the analysis and the options selected according to the state and type of analysis (Fig. 3).


**Fig. 3.** S.CORE: Analysis status page

*Problem.* How could the data embedded in a status page stay permanently updated without reloading the whole page?

*Motivation.* The status of the page has to be updated dynamically and permanently.

*Solution.* Periodically a request has to be executed to check if there is new data available. If so, this data has to be processed and the connected data element at the page updated or a new data element created.

*Examples.* (1) Flight status at Munich airport pages (www.munich-airport.de). (2) Goals during a soccer game at Spiegel online (www.spiegel.de).

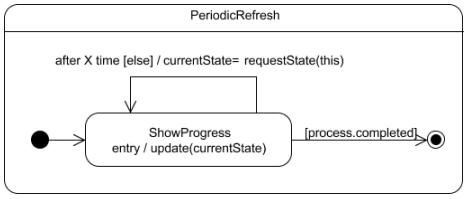


**Fig. 4.** RIA Pattern periodic refresh (UML state diagram)

## 4 Embedding RIA Patterns in Existing Web Methods

RIA patterns can be embedded in almost all existing methodologies to achieve considerable reduction of modeling and programming efforts. Our approach requires: (1) definition of extension points in the methodology for including references to RIA functionalities, (2) use of the state machines of the RIA patterns defined in Sect. 3, (3) definition of transformation rules that automatically integrate the behavior defined in the state machines into the models or code of the web application (for model-driven approaches only). Hence, a web developer using our approach only needs to indicate for which user interface (UI) objects of the web application a specific RIA pattern should be applied. He does not need to model the behavior of these UI objects as it is already specified by the state machines. He will afterwards use the state machines, integrating them in the models of the web application. If an automated generation process is followed, e.g. a model-driven approach, the developer will use a set of

model transformation rules that automatically add the corresponding behavior defined by the RIA pattern. This procedure and the advantages for the developer are made even clearer by the examples given in the next paragraphs. The use of state machines for the specification of the RIA patterns – a widely used modeling technique – makes the approach easily embeddable. In this paper and for demonstration purposes only, we use UWE as the hosting language, which is extended by RIA patterns.

**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 abstract and concrete presentation of a web application separately. The model-driven approach implemented by UWE is based on model transformations [1] that generate platform specific models from platform independent models and running programs based on these models [5]. UWE's outstanding feature is its reliance on standards: the modeling language is defined as a UML profile using the extension mechanisms provided by the UML. Model transformations are defined in ATL [5] and generate in the last step of the development process a JSF-based web application.

**RIA Patterns in the Design with UWE.** The objective of an extension covering a new concern is to augment the expressive power of the modeling language. The inherit risk is that successive extensions may end up with a powerful but not anymore intuitive language. Therefore, one of the main goals of UWE is minimalism and conciseness: web engineers should have to provide as much information as necessary for the generation of code, and as little as possible in order to keep diagrams readable.

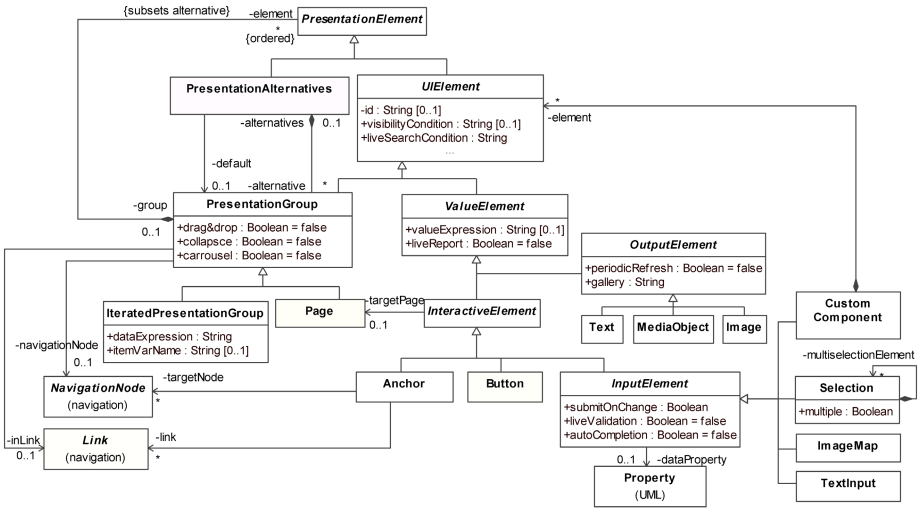


**Fig. 5.** Extended metamodel of the UWE Presentation Layer

In this sense, our RIA patterns are ideal constructs that provide a specification of the RIA behavior. The specification is given by a UML state diagram as shown in Sect. 3 for auto-completion and periodic refresh. The web engineer does not need to model this behavior each time to define such a RIA feature. It only remains to specify

for which UWE elements the auto-completion, the live validation, the period-refresh, etc. apply. This annotation is performed by tags added to existing model elements. A tag at model level implies the definition of a meta-attribute at metamodel level. Fig. 5 shows the metamodel for the presentation package of UWE, providing an overview of all meta-attributes defined so far in the UWE extension for RIAs. It shows how we embed RIA patterns in UWE.

The extension comprises: (1) the *liveSearchCondition* meta-attribute for *uiElement*, i.e. for all model elements that inherit form *uiElement*, (2) the meta-attributes *drag&drop, collapse* and *carrousel* for all *presentationGroups*, (3) *live-Report* for *value-Elements*, (4) meta-attributes *liveValidation* and *autoCompletion* for *inputElements*, (5) meta-attributes *periodicRefresh* and *gallery* for *outputElements*. In addition a new composition association was added to meta-class *Selection* providing the facility of an object to be on the one side multiple selectable and on the other side comprise a set of objects which are single selectable. Fig. 6 shows *auto-completion* of the image search and *periodic refresh* of the analysis status page.

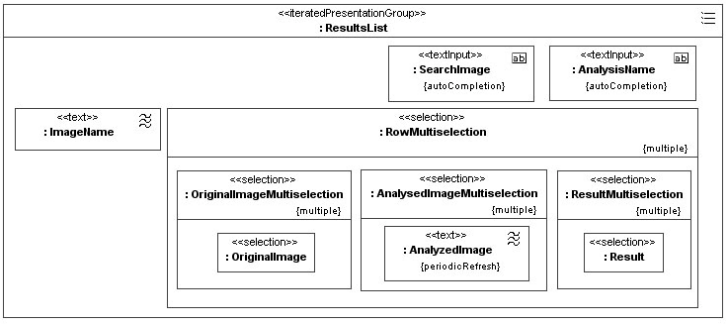


**Fig. 6.** S.CORE presentation model of analysis status page (excerpt)

## 5 Related Work

The need of an engineering support for RIA development has been recently addressed by several methods. New model-driven methods for designing RIAs are proposed by e.g. [2]. This method employs interaction spaces, tasks models and state machines. Disadvantage is that in case of reengineering web applications requires modeling from scratch. Several existing method have been extended for modeling RIAs. Toffetti et al. [13] focus on client or server side actions in data-intensive and collaborative RIAs describing events explicitly in WebML. In [14] issues related to behavior, single-page paradigm and content composition are treated extending OOHDM. UWE-R [6] is a light-weighted extension of UWE for RIAs, covering navigation, process and presentation aspects. In contrast to our work, UWE-R uses stereotypes for many of the extensions instead of meta-attributes. OOH4RIA [8] extends the OOH method introducing new model elements and applying to them new

transformations. These extensions do not focus on reuse and integration of RIA features in applications automatically.

Another type of approach combines modeling of a web application with an exiting method, such as in [10] where UWE is complemented with the RUX method for the UI design. The approach consists of the transformation of UWE presentation model to a RUX abstract interface model (AIM), which is afterwards enriched with typical RIA user interface actions. In this approach RIA features are introduced into models at a lower level of abstraction than in the current approach. Moreover, RIA requirements and code library based on best practices are also available, like [12].

## 6 Conclusions and Future Work

We have presented patterns in the form of UML state machines for modeling RIAs. Our approach is general as it can be used on top of virtually every UML conform language. Future work based on our patterns includes an extension of the modeling tool for UWE, MagicUWE (uwe.pst.ifi.lmu.de/toolMagicUWE.html), to support the notation and the integration of the RIA patterns. Furthermore, we plan to extend our work on validating web design models by model checking [3] to cover our patterns, too. The implementation of our real world case study S.CORE was performed by manual translation of the RIA patterns to code. We also plan to implement model-driven code generation out of our patterns using the JSF framework [5].

## References

[1] ATLAS Transformation Language & Tool, http://www.eclipse.org/m2m/atl/doc/

[2] Dolog, P., Stage, J.: Designing Interaction Spaces for Rich Internet Applications with UML. In: Baresi, L., Fraternali, P., Houben, G.-J. (eds.) ICWE 2007. LNCS, vol. 4607, pp. 358–363. Springer, Heidelberg (2007)

[3] Knapp, A., Zhang, G.: Model Transformations for Integrating and Validating Web Application Models. In: Proc. MOD 2006, LNI P-82, pp. 115–128, GI (2006)

[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 (12), ch. 7, vol. 12, pp. 157–191. Springer, Heidelberg (2008)

[5] Kroiss, C.: Model-based Generation of Web Applications with UWE Diploma thesis (in German). LMU (2008)

[6] Machado, L., Filho, O., Ribeiro, J.: UWER: uma extensão de metodologia em Engenharia Web para Rich Internet Applications. II Simpósio de Informática da PUCRS, RS. Hifen Magazine 32(62), 205–212 (2008)

[7] Mahemoff, M.: Ajax Design Patterns. O'Reilly, Sebastopol (2006)

[8] Meliá, S., Gómez, J., Pérez, S., Díaz, O.: A Model-Driven Development for GWT-based RichInternet Applications with OOH4RIA. In: Proc. of ICWE 2008, pp. 13–23. IEEE, New York (2008)

[9] Morozova, T.: Modeling and Generating Web 2.0 Applications. Diploma thesis (in German). LMU (2008)

[10] Preciado, J.C., Linaje, M., Morales, R., Sánchez-Figueroa, F., Zhang, G., Kroiss, C., Koch, N.: Designing Rich Internet Applications Combining UWE and RUX-Method. In: Proc. of ICWE 2008, pp. 148–154. IEEE, New York (2008)
[11] S.CO LifeScience, http://www.sco-lifescience.de/ (Last visited 17.04.2009)
[12] Scott, B.: RIA Patterns. Best Practices for Common Patterns of Rich Interaction, http://www.uxmatters.com/mt/archives/2007/03/ (Last visited 10-02-2009)
[13] Toffetti, G., Comai, S., Bozzon, A., Fraternali, P.: Modeling Distributed Events in Data-Intensive Rich Internet Applications. In: Proc. of ICWE 2007. LNCS, vol. 4607, pp. 593–602. Springer, Heidelberg (2007)
[14] Urbieta, M., Rossi, G., Ginzburg, J., Schwabe, D.: Designing the Interface of Rich Internet Applications. In: Proc. of LA-Web 2007, pp. 144–153. IEEE, Los Alamitos (2007)