Modernization of Legacy Web Applications into Rich Internet Applications*

Roberto Rodríguez-Echeverría, José María Conejero, Pedro J. Clemente, Juan C. Preciado, and Fernando Sánchez-Figueroa

University of Extremadura Spain,
Quercus Software Engineering Group
{rre,chemacm,pjclemente,jcpreciado,fernando}@unex.es
http://quercusseg.unex.es

Abstract. In the last years one of the main concerns of the software industry has been to reengineer their legacy Web Applications (WAs) to take advantage of the benefits introduced by Rich Internet Applications (RIAs), such as enhanced user interaction and network bandwith optimization. However, those reengineering processes have been traditionally performed in an ad-hoc manner, resulting in very expensive and error-prone projects. This situation is partly motivated by the fact that most of the legacy WAs were developed before Model-Driven Development (MDD) approaches became mainstream. Then maintenance activities of those legacy WAs have not been yet incorporated to a MDA development lifecycle. OMG Architecture Driven Modernization (ADM) advocates for applying MDD principles to formalize and standardize those reengineering processes with modernization purposes. In this paper we outline an ADM-based WA-to-RIA modernization process, highlighting the special characteristics of this modernization scenario.

Keywords: Web Models Transformations, Software Modernization, Software Reengineering, Rich Internet Applications.

1 Introduction

Rich Internet Applications (RIAs) have emerged as the most promising platform for Web 2.0 development by the combination of the lightweight distribution architecture of the Web with the interface interactivity and computation power of desktop applications, with benefits on all the elements of a WA (data, business logic, communication, and presentation). Among others, RIAs offer online and offline capabilities, sophisticated user interfaces, the possibility to store and process data directly on the client side; they offer high levels of user interaction, usability and personalization. RIAs also minimize bandwidth usage, and separate presentation and content at the client side [16].

* This work has been supported by MEC (TIN2008-02985), FEDER and Junta Extremadura.

A. Harth and N. Koch (Eds.): ICWE 2011 Workshops, LNCS 7059, pp. 236–250, 2011.
© Springer-Verlag Berlin Heidelberg 2011

To take advantage of these new capabilities, the industry is performing a reengineering of their legacy WAs to produce RIA clients. Unfortunately, a huge number of those legacy WAs were developed before most promising Model-Driven Web Engineering (MDWE) [18] methodologies were mature enough for mainstream. Then the industry posses a wide catalogue of complex WAs that were developed without following any MDD principle or technique. And the maintenance activities of those legacy WAs cannot be incorporated to the MDA development lifecycle of a company. Among other negative consequences, a lot of those legacy applications may lack a comprehensive up-to-date documentation and they may have been poorly maintained integrating new technologies without a defined strategy. In this complex scenario, the industry demands formalization and standardization of reengineering processes to reduce the expensive costs and high risks introduced by ad-hoc reengineering processes. In this setting, OMG Architecture-Driven Modernization (ADM) advocates for the application of Model-Driven Development (MDD) techniques and tools to formalize and standardize software reengineering processes.

Precisely, the major objective of the work presented in this paper is to define a flexible framework for the systematic and semi-automatic modernization of legacy non-model-based data-driven WAs into RIAs following OMG ADM principles. This paper shows then part of our work performed inside a complete modernization project we are currently developing in partnership with a national software company. In concrete, in this paper we only present a brief outline of our process, focusing specially on the RIA pattern identification activity. We conceived WA-to-RIA modernization as the process of building a RIA client from the legacy WA presentation and navigation layers and the required service-oriented connection layer with the underlying business logic at server side. Besides, we consider a RIA client is characterized for satisfying a set (or subset) of RIA features (presented in Section 2).

According to the OMG Architecture-Driven Modernization (ADM), the main objectives of our work may be summarized as follows:

- Legacy WA knowledge discovery. Our framework tries to define which information from the legacy system should be of interest for the modernization. And it tries to refine the knowledge extracted to alleviate the modernization costs. The acquired knowledge could be very heterogeneous covering aspects from technical (e.g. components, flow controls, etc.) to business domains (e.g. tasks, business rules, etc.).
- Target architecture definition. Recently, many approaches have appeared [6][14][22] in the Web Engineering community for the definition of RIA architectures. We try to apply the results of those proposals in our intent to derive a conceptual description of RIAs, which was independent of any particular technological platform and was useful on modernization processes.
- Transformation steps from the original system to the target one. Our framework tries to define the necessary sequence of steps to transform a legacy WA into a RIA, keeping the required flexibility to cope with different modernization scenarios.

The rest of the paper is structured as follows. Section 2 presents the collection of RIA features we consider to define the RIA client concept. Section 3 defines the system we use to illustrate our approach. Section 4 introduces our approach. The related work is commented in Section 5. Finally, main conclusions and future work are presented in Section 6.

2 Main Features of RIAs

In order to give a definition to the RIA client concept and to identify the relevant information to extract from the legacy WA for modernization, we have performed a deep analysis of the RIA-extended MDWE approaches and collected all the RIA features covered by them from a conceptual point of view. We have performed the collection and annotation of RIA features in a high level of abstraction, trying to avoid low level or technological concerns, and trying to provide a unified vision of them.

We consider the work in [16] as the starting point in the evolution and extension of a set of MDWE approaches in order to fulfill the new expressivity requirements introduced by the RIA development. According to that decision, we have only considered works published since 2005. Among the different proposals available in literature, we have studied both the most mature ones and also the recent proposals under development that may have some impact in the next future: WebML4RIA [6], OOHDM-RIA [19], OOH4RIA [11], UWE for RIA [7], RUX-Method [9], UWE-R [10], OOWS 2.0 [22], ADRIA [5] and IAML [24].

Following we present the collecion of RIA features we have:

- RF01. Data storage on client side. This feature refers to the capability of the client side to store data in a volatile or persistent way. The persistent data storage on client side is becoming a clear trend for current RIAs (key feature of HTML 5 standard).
- RF02. Multiple data sources or types. Actual RIAs can connect to different data providers (databases, Web services, Web APIs, etc.) and use different data formats (raw datasets, XML, JSON, etc.).
- RF03. Multimedia and Animation Support. Temporal Behavior. This feature refers to the capability of the client side to manage complex animations and multimedia content properly in order to enhance the user interaction.
- RF04. Logic execution on client side. It refers to the capability of the client side to execute part of the business logic inside its own runtime. Together with RF01 may reduce considerably server roundtrips and enhance user experience and productivity.
- RF05. Multithreading or concurrency. This feature refers to the capability of the client side to launch simultaneously different functionality threads. Its most widespread use is the ability of the client side of keeping a responsive interface while requesting data from the server side.
- RF06. Multidevice User Interface. This feature refers to the capability of a RIA to be accessed from a wide range of heterogeneous client terminals (user agents or devices). In the last years, RIAs have spreaded to the mobile

market and they have become one of the most preferred approaches to deploy applications because of their independence of technology.

- RF07. Single-Page Paradigm or Partial Page Refresh. This feature refers to the capability of a RIA client to present a desktop-like user interface avoiding the Click-Wait-and-Refresh-Cycle, characteristic of Web clients. This feature could be seen as a consequence of RF04 and RF12, at least.

- RF08. Rich UI Components (widgets). This feature refers to the capability of a RIA client to use a whole constellation of interactive and complex controls and components for UI composition. It supposes logic execution on client side.

- RF09. Rich User Interaction. This feature refers to the capability of a RIA client to define enhanced interactions and complex UI behaviours by explicitiing orchestrations among widgets and server-side logic.

- RF10. Client runtime control. This feature refers to the capability of a RIA client to use and control partially the functionality of its runtime and to change its default behaviour, e.g. the back button of a Web browser.

- RF11. Communication started on server side (push model). This feature refers to the capability of a RIA to overcome the request-response communication model of Web applications. In a RIA both tiers (client and server) can innitiate a communication process with the other one.

- RF12. Asynchronous communication. This feature refers to the capability of a RIA client to send a request to the server without blocking until a response is sent back. A RIA can keep working normally and handle the response when necessary.

- RF13. Bulk data client-to-server transfers. It refers to the capability of a RIA client to send a collection of data to the server at once to reduce the server roundtrips. A RIA client stores collections of related data produced by the normal execution process and, at a given time, it sends the whole set of data to the server at once.

- RF14. Synchronization between client and server tiers. This feature refers to the capability of a RIA to keep data consistency among the different tiers of the application. This is a high level feature and then it can be decomposed in communication sequences between tiers. This feature could be seen as a combination of RF01, RF04, RF13, and a concrete synchronization policy.

- RF15. Offline mode. This feature refers to the capability of a RIA client to change seamlessly its operation mode between standalone, without live connection to the server, and online modes, with connection to the server. This is also a high level feature (referred as an architectural feature in some works) and then it can be seen as a consequence of the application of RF01, RF04, RF07 and RF14, at least.

3 Illustrative Example

In order to illustrate the main steps of our approach, let us consider JAVA Pet Store[1] Demo (Petstore) as our legacy WA. Petstore 1.3.2 was built on 2003 by JAVA BluePrints team to exemplify the development of a WA by means of the J2EE SDK technologies. Following we present the main reasons to select this sample legacy WA.

- The source code is publicly available. And it is a medium-size system.
- There exists a comprehensive documentation because it is conceived as a training project.
- It could be considered a well-known sample application and the baseline code of many WAs developed during those years.
- Its development is based on the BluePrints Web Application Framework (WAF), which inspired next JAVA Web application frameworks. So it presents the main elements of current MVC-based Web Application Frameworks.
- We think it is representative enough to illustrate the main points of the proposed approach.
- Additionally, Petstore has been evolving along with the JEE SDK to illustrate the features of the new versions. So, with the release of JEE 5 SDK, Petstore was reengineered to illustrate how the Java EE 5 platform can be used to develop an AJAX-enabled Web 2.0 application.

In this setting, Petstore WA is perfectly suitable to be used as our case study since the original application is used as the input for our approach whilst the new Petstore 2.0 WA represents the desired output of the approach.

Basically, Petstore WA provides customers with online shopping. Through a Web browser, a customer can browse the catalog, place items to purchase into a virtual shopping cart, create and sign in to a user account, and purchase the shopping cart contents by placing an order with a credit card.

As one of our main goals consists on modernizing the presentation tier, we have focused on the catalog functionality of the Storefront component. In concrete, we are interested in product and item page shown in figure 1. These two web pages are dinamycally generated from application data. The former displays a product listing (all items of Chihuahua product in figure). The latter shows the details of a concrete item (Adult Male Chihuahua in figure). Every product item of the product page links with its corresponding item page. Clearly a Master/Detail relationship is set between the main data displayed by both pages.

From a RIA viewpoint, contrary to the multipage solution presented by Petstore 1.3.2, this scenario of data relationship is realized in a single page by applying the Master/Detail screen pattern [20]. This is an ideal pattern for creating an efficient user experience by allowing the user to stay in the same screen while navigating between items. Moreover, as figure 2 illustrates, Master/Detail screen pattern is the solution adopted by the Petstore 2.0[2] implementation for

[1] Version 1.3.2: [Aug 04, 2003]
http://java.sun.com/blueprints/code/jps132/docs/index.html

[2] http://java.sun.com/developer/technicalArticles/J2EE/petstore/

Fig. 1. Master/Detail relationship between product and item pages

Fig. 2. Master/Detail Screen pattern merging legacy product and item pages

the scenario depicted above. This modernization scenario is precisely the case study we have selected to illustrate our approach.

4 The Approach

As aforementioned, the approach presented in this work briefly introduces and outlines our modernization process of a legacy WA. As figure 3 shows, the main objective of our process is to generate a RIA client of the legacy WA and the necessary service-oriented connection layer with the underlying business logic. The RIA client could be composed of a rich UI (highly interactive), the data stored at client side, the logic processed at client side, and the infrastructure logic for server communication and synchronization. Most of the server-side code would remain unmodified so the system could keep working as a WA. With that purpose, a connection layer would be built between the new RIA client and the original business logic. This layer aims to cope with the derived data and logic distribution concerns and seamlessly integrate an asynchronous communication model between client and server.

As depicted in figure 4, our modernization process consists on 5 main phases: (1) static and dynamic information extraction from the source (data, logic and presentation) and the configuration files of the original WA; (2) knowledge representation and refinement on a technology independent language (we use KDM),

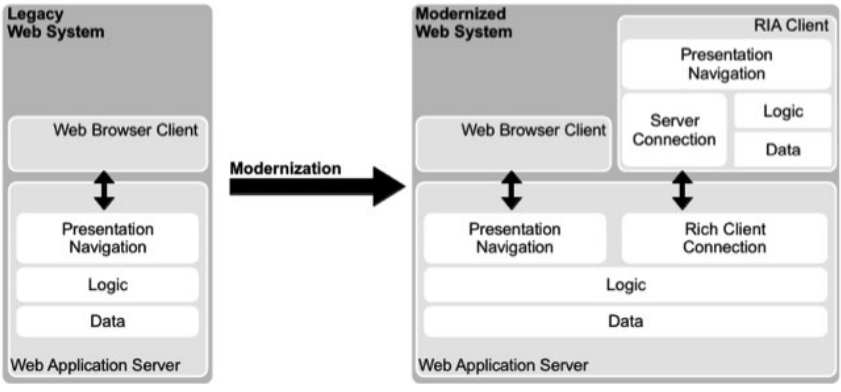

Fig. 3. Target system architecture

the extracted information is incorporated in a model with a higher level of abstraction; (3) optional projection of the conceptual system to a specific RIA-extended MDWE approach; (4) optional Web models refinement by applying RIA patterns at more concrete level; and (5) final code generation.

According the ADM horseshoe model proposed by OMG [21], which defines 3 modernization domain levels (technology, application & data, and business), we argue our approach would be located at the second domain level (application and data architecture) because we think it involves major changes (beyond technical domain) that clearly affect the application architecture. Fundamentally, these changes are related to:

- The UI structure and organization (RF06-RF08). The multipage structure of the legacy WA presentation layer should be frequently modified according to the single-page paradigm (RF07) characteristic of RIA clients. A componentization process should be performed to map plain HTML display elements and controls into RIA widgets. And all the main elements of the legacy UI layout should be rearranged according to the new paradigm.
- The UI control flow (RF05, RF09-RF10). The hyperlink-based interaction model should be transformed to an event-based interaction model. In a RIA client navigation is not conceived as a sequence of hyperlinked page. Navigation is realized as a sequence of UI state transitions driven by events. Some of those transitions could not imply a request to the server. An UI state transition could be basically defined as an update (screen update) of the current UI components or as a new components load (new screen load).
- The client to server communication (RF11-RF14). RIA clients frequently interact with server logic following a service-oriented model, which may imply major changes on server side logic interface. Additionally, single-page RIA clients require an asynchronous communication model to maintain a responsive UI.

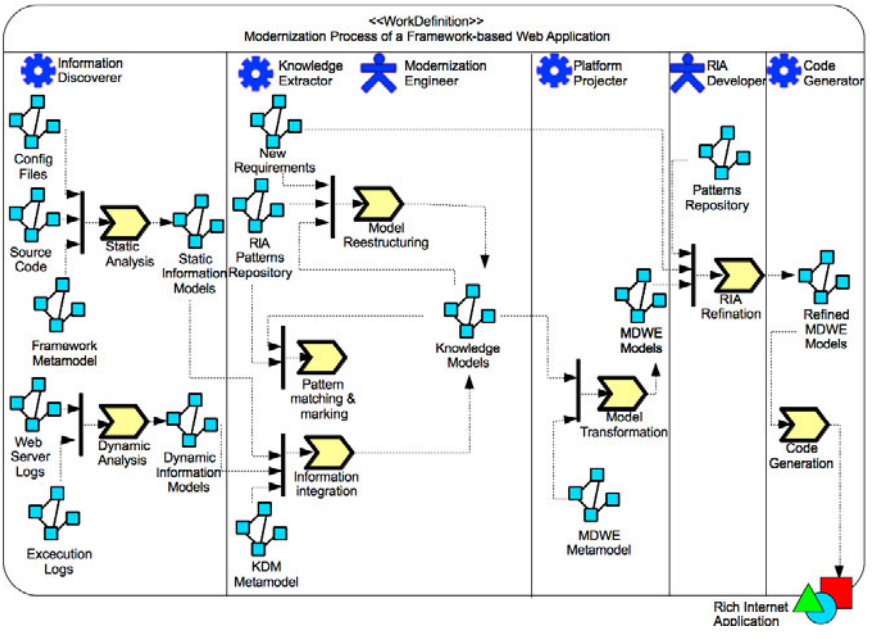

Fig. 4. Modernization process overview

– The offline work mode (RF15). Many RIA clients provide their users with the capability of switching between online and offline modes. Obviously, this is a highly device-dependant feature, because at client side: data should be stored (RF01); and business logic should be executed (RF04).

Following, we try to provide a more detailed vision of the main stages of our modernization process, using the Petstore sample WA to illustrate them. We will take special attention to the RIA pattern recognition step in phase 2.

4.1 Information Extraction and Representation

As shown in figure 4, the first phase of our process tries to reduce the complexity of the modernization process by switching from the heterogeneous world of implementation technologies to the homogeneous world of models. For this purpose, following ADM recommendations, we have used available solutions for code to model transformation (static analysis), such as MoDisCo discoverers and metamodels for Java, JSP and XML (Specific Abstract Syntax Tree Metamodels, SASTM[3]). Language-dependent models representing the whole legacy WA are then obtained as final products of this phase. Then, the process provides us, thus, with the ability of working directly with models since this moment. Further we have considered the convenience of specifying additional metamodels to

[3] http://www.omg.org/spec/ASTM/

capture supplemental information from the legacy WA, e.g. WAF information. But we decided to postpone that goal to the next iteration of our approach.

According to our goal of generating a RIA client, the most relevant information to extract accurately is the one involved in the following concerns:

- UI Layout. Commonly, legacy WAs have been built to keep a uniform UI structure and organization to increase usability and to present a recognizable look&feel. To capture that UI Layout is then a preponderant requirement of our modernization process. So the legacy WA look&feel could be regenerated in the RIA client. That is a difficult task. Our approach consists on extracting that kind of information from the template system used by the Web application framework. In the concrete case of our example, we get some basic UI Layout information from the configuration file of the template system. Figure 5 (left side) shows the UI Layout of the item page.
- Web page and data relationships. Dynamic Web pages are generated on the fly to display different values of the application data. So every dynamic Web page defines a concrete view of application data. The correct specification of those views and the related data entities are key to infer the proper componentization of the legacy WA UI. Figure 5 (right side) shows the JSP code excerpt that relates the item page to the item data.
- Navigational map. In order to generate a RIA client according to the single page paradigm (RF07) is necessary to extract and process the whole navigational map of the legacy WA. So grouping and clustering activities could be performed to assist on the componentization process. In our example, the navigation information could be extracted from different sources: JSP pages, template system configuration file, request mapping file (concrete responses to client request may be specified) and Java code, indeed. For this work, we are only considering the navigational map extracted from the JSP pages.
- Operational map. This map is a subset of the navigational map concerning only the requests dispatched by the controller component of the legacy WA as action calls to the business layer. The operational map is useful to discover the request flows between client and server tiers (communication model) and to identify the operations performed over the data. In our example, the operational map could be extracted from JSP pages and the request mapping file.

```
<screen name="item">
 <parameter key="title" value="Item" direct="true"/>
 <parameter key="banner" value="/banner.jsp" />
 <parameter key="sidebar" value="/sidebar.jsp" />
 <parameter key="body" value="/item.jsp" />
 <parameter key="mylist" value="/mylist.jsp" />
 <parameter key="footer" value="/footer.jsp" />
</screen>
```

```
<c:set value="${catalog.item}" var="item" />

<p class="petstore_title">
 <c:out value="${item.attribute}"/>
 <c:out value="${item.productName}"/>
</p>
```

Fig. 5. Legacy code

Finally, we introduce an activity to analyze dynamically the legacy WA. We argue that the analysis of the runtime traces could provide us with valuable information about user interaction. That interactivity information could drive modernization decisions to take in following phases of the process.

4.2 Knowledge Inference and Representation

This is the main phase of the modernization process. The goal of this phase would be to derive an enriched conceptual specification of the legacy system in a technology-independent model (knowledge model) from the information stored inside the static and dynamic models generated on the previous phase. Moreover, the knowledge model will be continuously refined according to the modernization goals. From an overall viewpoint, this phase is composed of three fundamental steps:

1. Transformation of the intermediate static models (SASTM) onto the technology-independent knowledge model (KDM[4]), integrating all the extracted information. ADM suggests to use a M2M transformation to perform this step, as [15][4][3] exemplified.
2. Enrichment of the KDM models from the dynamic information obtained.
3. Intermediate model refinement by finding expressions of characteristic RIA patterns.

On one hand, figure 6 shows an excerpt of the KDM representation (simplified) of the Petstore sample WA, as an example of the output of the first step of this phase. As shown, we are only considering UI and Code KDM Packages. In the UI Packages JSP pages are modeled as instances of the *Screen* metaclass, product and item *Screen* instances in the figure. Both *Screen* instances are related by a *UIFlow* instance that represents a navigation flow from the product *Screen* instance to the item *Screen* instance. We are considering only the main area of the JSP pages. So both *Screen* instances are only composed of *UIField* instances representing the data to be displayed. Every *UIField* is related with a *Member Unit* instance by a *Display* relationship. In this case, the *Member Unit* instances represent members of the item instance of *Class Unit*. That way both screens specify a different view of the item *Class Unit*. Additionally (not shown in the figure), the product *Screen* instance actually displays a collection of instances of item *Class Unit*.

On the other hand, the model refinement step is performed in two sequential activities: (1) identification of pattern expressions in the knowledge model; and (2) restructuring of the knowledge model according to the patterns identified.

First of all, we try to refine the model by locating automatically RIA pattern expressions in the knowledge model. This activity is performed by a pattern matching process. Selected RIA patterns stored in the repository are processed sequentially. Marks are introduced in the model knowledge to signal pattern

[4] http://www.omg.org/spec/KDM/1.1/

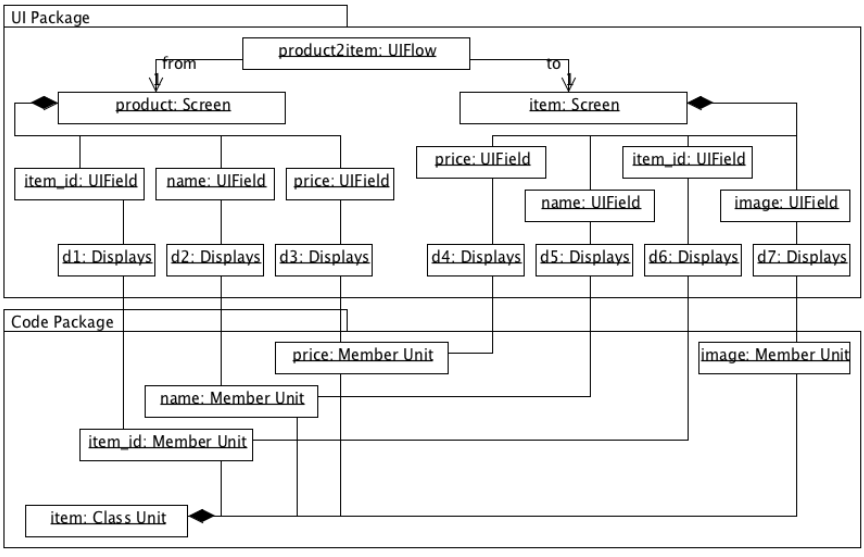

Fig. 6. Mater/Detail Screen Pattern in KDM (simplification)

identifications. As mentioned in section 3, the illustrative scenario we are considering is the detection of the Master/Detail screen pattern. In this case, basically, we try to locate two instances of the *kdm::ui::Screen* metaclass related in a sequential flow and displaying the same data entity but at different level of detail. In concrete, the master page will display less data than the detail page. This scenario is precisely the situation depicted in figure 6. To automatize this activity we are trying to use the QVT[5] language. Current results are promising, but the precision of the pattern matching process is still low. One reason of this low precision could be related with the high level of abstraction of the KDM UI Package which may lack necessary elements to represent Web user interfaces as [2] suggests. Probably we should also review the way we are using KDM. Another reason could be the lack of contextual information that could lead to detect false positives. To alleviate that situation we think it would be necessary to get additional information to semantically define both Web pages (and the data displayed) and their relationship within the whole system.

After pattern recognition and signalling, the knowledge model is ready to be restructured according to the patterns identified. We considered this activity requires human intervention. The modernization engineer should review all the marks introduced in the knowledge model and select one of the available restructurings for each one, keeping a valid model. Returning to our example, the engineer could select between 2 restructuring choices: (1) applying the Master/Detail Screen pattern as mentioned in section 3; or (2) applying the Quicklook pattern

[5] http://www.omg.org/spec/QVT/

Fig. 7. Quicklook pattern in the search results page

(hover text) as the search page of Petstore 2.0 does to present details of the search results (figure 7).

4.3 Platform Projection

We have decided to introduce an optional step previous to the generation of the final code of the RIA client. This step consists on projecting the refined knowledge models of the legacy WA into RIA-extended MDWE models. Current techniques and tools of M2M transformation could assist on this projection. We consider this optional step could provide the engineer with the following advantages:

- The target system would be specified in a language nearer to Web and RIA domains. So it could be processed at a fine-grained level.
- Some tool or tool chain to support the development of the system, e.g. WebRatio for WebML [1], would assist the engineer in the modernization process or future maintenance activities.
- A repository of patterns could be used to leverage the stored know-how on system refinement [17], and to detect potential problems generated during the modernization process.
- A code generation engine that appears as a fundamental requirement for the forward engineering stage of the modernization process.

4.4 Code Generation

Following our idea of reusing MDD techniques and tools, final code generation could be performed by means of available code generation engines. On one hand, for instance, to generate the client side we could use the generation engines of the toolkits of main MDWE-RIA approaches, such as WebRatio and RUX-Tool [8]. On another hand, for server side connection layer generation we should evaluate the application of different model to code transformation tools, such as OMG MOF Model to Text, JET, Xpand, etc.

5 Related Work

Due to the wide scope and complexity of the process presented here, there is a high number of related approaches and they are really heterogeneous. This section points out some of these works as example of this heterogeneity.

During the last decade, as stated in [13], important works in the reverse engineering domain have been developed. VAQUISTA [23] proposes the utilization of different reverse engineering techniques to make the migration of the user interface of a WA to different platforms easer. Similarly, the work in [12] applies reverse engineering techniques to migrate a multipage interface of a WA to a single page interface (Web 2.0). All these approaches are closely related to the reverse engineering phase of the modernization process presented here.

[19]and [17] propose approaches to systematically incorporate RIA features into legacy WAs. However, contrary to the work presented here, these approaches are applied to legacy WAs that were developed by using MDD techniques and methodologies.

On the other hand, in the last years there have appeared some approaches to the application of MDD principles and techniques for the maintainability of software systems, e.g. in activities of software migration or modernization. In that sense, MoDisco [3] is a generic, extensible and open source approach for software modernization that makes an intensive use of MDD principles and techniques. Our work presents a specialization of the framework defined by MoDisco to be applied in concrete modernization scenarios from legacy WAs into RIAs.

6 Conclusions and Future Work

This work present an outline of our approach for the definition of a systematic process for WA-to-RIA modernization, by applying MDE principles, techniques and tools. One main requirement of this process is to make an extensive use of ADM related specifications. In concrete, the main goal of the modernization process presented consists on generating a RIA client from the legacy WA presentation and navigation layers and its corresponding service-oriented connection layer with the underlying business logic at server side. We have specially focused on the RIA pattern identification activity. Master/Detail Screen pattern and Quicklook pattern has been proposed as possible solutions for multipage master/detail relationships on the legacy WA.

Moreover, this work also depicts a collection of essential RIA features we have collected to understand the concept of RIA client. On one hand, these features provided us with the necessary information to define a conceptual modernization architecture in KDM. On another hand, they helped us on the specification of the kind of information we should extract from the legay WA in order to perform its RIA modernization.

Regarding the tool support, we are currently involved in the definition of our tool chain to systematize the modernization process by assisting the engineer team in the many and complex tasks to accomplish. For the reverse engineering

phase, we are evaluating the possibility of adopting MoDisco as tool framework. Meanwhile, for the forward engineering phase, we may use mainstream MDWE methods and tools, e.g. WebRatio and RUX-Tool.

Given the extension and complexity of every modernization process and the initial stage of our approach, we have a great amount of related researching lines to follow. Among them, we are principally interested in 3: (1) extracting more accurate dynamic information (interaction models) from the legacy WA in order to infere the necessary knowledge to drive the data and logic distribution between client and server sides; (2) extending the application of ADM specifications to the whole process and considering business domain modernization; and (3) integrating properly the modernization tool chain to reduce costs and to leverage modernization knowledge reuse. Additionally, to confirm our RIA features relevance we will try to validate them with practitioners.

References

1. Acerbis, R., Bongio, A., Brambilla, M., Butti, S., Ceri, S., Fraternali, P.: Web Applications Design and Development with WebML and WebRatio 5.0. In: Bertrand Aalst, W., Mylopoulos, J., Sadeh, N.M., Shaw, M.J., Szyperski, C., Paige, R.F., Meyer (eds.) TOOLS EUROPE 2008. LNBIP, vol. 11, pp. 392–411. Springer, Heidelberg (2008)
2. Barbier, F., Deltombe, G., Parisy, O., Youbi, K.: Model Driven Reverse Engineering: Increasing Legacy Technology Independence. In: Second India Workshop on Reverse Engineering, Thiruvanantpuram (2011)
3. Bruneliere, H., Cabot, J., Jouault, F.: MoDisco: A Generic And Extensible Framework For Model Driven Reverse Engineering. In: IEEE/ACM International Conference on Automated Software Engineering, pp. 1–2 (2010)
4. Izquierdo, J.L.C., Molina, J.G.: An Architecture-Driven Modernization Tool for Calculating Metrics. IEEE Software 27(4), 37–43 (2010)
5. 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)
6. Fraternali, P., Comai, S., Bozzon, A., Carughi, G.T.: Engineering rich internet applications with a model-driven approach. ACM Transactions on the Web 4(2), 1–47 (2010)
7. Koch, N., Pigerl, M., Zhang, G., Morozova, T.: Patterns for the Model-Based Development of RIAs. In: Gaedke, M., Grossniklaus, M., Díaz, O. (eds.) ICWE 2009. LNCS, vol. 5648, pp. 283–291. Springer, Heidelberg (2009)
8. Linaje, M., Preciado, J.C., Morales-Chaparro, R., Rodríguez-Echeverría, R., Sánchez-Figueroa, F.: Automatic Generation of RIAs Using RUX-Tool and Webratio. In: Gaedke, M., Grossniklaus, M., Díaz, O. (eds.) ICWE 2009. LNCS, vol. 5648, pp. 501–504. Springer, Heidelberg (2009)
9. Linaje, M., Preciado, J.C., Sanchez-Figueroa, F.: Engineering Rich Internet Application User Interfaces over Legacy Web Models. IEEE Internet Computing 11(6), 53–59 (2007)
10. Machado, L., Filho, O., Ribeiro, J.: UWE-R: an extension to a web engineering methodology for rich internet applications. WSEAS Transactions on Information Science and Applications 6(4), 9 (2009)

11. Meliá, S., Gómez, J., Pérez, S., Díaz, O.: A Model-Driven Development for GWT-Based Rich Internet Applications with OOH4RIA. In: 2008 Eighth International Conference on Web Engineering, pp. 13–23 (July 2008)
12. Mesbah, A., van Deursen, A.: Migrating Multi-page Web Applications to Single-page AJAX Interfaces. In: 11th European Conference on Software Maintenance and Reengineering (CSMR 2007), pp. 181–190 (March 2007)
13. Patel, R., Coenen, F., Martin, R., Archer, L.: Reverse Engineering of Web Applications: A Technical Review. Technical Report July 2007, University of Liverpool Department of Computer Science, Liverpool (2007)
14. Pérez, S., Díaz, O., Meliá, S., Gómez, J.: Facing Interaction-Rich RIAs: The Orchestration Model. In: 2008 Eighth International Conference on Web Engineering, pp. 24–37 (July 2008)
15. Pérez-Castillo, R., De Guzmán, I.G.-R., Piattini, M.: Business Process Archeology using MARBLE. In: Information and Software Technology (2011)
16. Preciado, J.C., Linaje, M., Sanchez, F., Comai, S.: Necessity of methodologies to model Rich Internet Applications. In: Seventh IEEE International Symposium on Web Site Evolution (2005)
17. Rodríguez-Echeverría, R., Conejero, J.M., Linaje, M., Preciado, J.C., Sánchez-Figueroa, F.: Re-engineering legacy Web applications into Rich Internet Applications. In: 10th International Conference on Web Engineering (2010)
18. Rossi, G., Pastor, O., Schwabe, D., Olsina, L.: Web Engineering: Modelling and Implementing Web Applications. Human-Computer Interaction Series (October 2007)
19. Rossi, G., Urbieta, M., Ginzburg, J., Distante, D., Garrido, A.: Refactoring to Rich Internet Applications. A Model-Driven Approach. In: 2008 Eighth International Conference on Web Engineering, pp. 1–12 (July 2008)
20. Scott, B., Neil, T.: Designing Web Interfaces: Principles and Patterns for Rich Interactions. O'Reilly Media (2009)
21. Ulrich, W.: Modernization Standards Roadmap, pp. 46–64 (2010)
22. Valverde, F., Pastor, O.: Facing the Technological Challenges of Web 2.0: A RIA Model-Driven Engineering Approach. In: Vossen, G., Long, D.D.E., Yu, J.X. (eds.) WISE 2009. LNCS, vol. 5802, pp. 131–144. Springer, Heidelberg (2009)
23. Vanderdonckt, J., Bouillon, L., Souchon, N.: Flexible reverse engineering of web pages with VAQUISTA. In: Proceedings Eighth Working Conference on Reverse Engineering, pp. 241–248 (2001)
24. Wright, J.M.: A Modelling Language for Interactive Web Applications. In: 2009 IEEE/ACM International Conference on Automated Software Engineering, pp. 689–692 (November 2009)