Model Driven Extraction of the Navigational Concern of Legacy Web Applications*

Roberto Rodríguez-Echeverría, José M. Conejero, Pedro J. Clemente, Víctor M. Pavón, and Fernando Sánchez-Figueroa

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

Abstract. Nowadays, there is a current trend in software industry to modernize traditional Web Applications (WAs) to Rich Internet Applications (RIAs). RIAs improve the user experience by combining the lightweight distribution architecture of the Web with the interface interactivity and computation power of desktop applications. In this context, Model Driven Web Engineering (MDWE) approaches have been extended with new modeling primitives to obtain the benefits provided by RIA features. However, during the last decade, widespread language-specific web frameworks have supported actual web system development. In this paper we present a model driven modernization process to obtain RIAs from legacy web systems based on such frameworks. model driven techniques reduce complexity and improve reusability of the process, making the development more systematic and less error prone. Being navigational information of upmost importance for the modernization process of a web application, the paper is focused on presenting the model driven extraction of such concern from the legacy system artifact, presenting the extraction tools and process.

Keywords: Model-driven Engineering, Re-engineering, Web Applications, RIA.

1 Introduction

Rich Internet Applications (RIAs) have emerged as the most promising platform for Web 2.0 development combining the lightweight distribution architecture of the web with the interface interactivity and computation power of desktop applications [11]. To take advantages of these new capabilities, there is a current trend in the industry to perform a modernization of their legacy WA to produce RIA counterparts. This trend is, even, more evident with the transition to the

* Work funded by Spanish Contract MIGRARIA - TIN2011-27340 at Ministerio de Ciencia e Innovación and Gobierno de Extremadura (GR-10129) and European Regional Development Fund (ERDF).

M. Grossniklaus and M. Wimmer (Eds.): ICWE 2012 Workshops, LNCS 7703, pp. 56–70, 2012.
© Springer-Verlag Berlin Heidelberg 2012

forthcoming HTML5 that implements natively most of these features gaining momentum.

In this context, MDWE approaches [14] have been extended with new modeling primitives to obtain the benefits provided by RIA features [7][10][17]. This way, introducing RIA features in legacy WA developed using models becomes a feasible task as it has been shown in [15][13]. However, during the last decade, widespread language-specific Web frameworks (e.g. Struts[1]) have supported the actual developments of these WAs, neglecting the benefits provided by model driven approaches. These frameworks are often tied to the programming-language level, making maintenance and modernization processes a difficult task. Traditionally, these modernization processes have been performed in an ad-hoc manner, resulting in very expensive and error-prone projects.

This work is part of a larger research project, called MIGRARIA, where a systematic and semi-automatic process to modernize legacy non-model-based data-driven WAs into RIAs has been defined. The process is based on model driven reengineering techniques used to mainly obtain (i) a new RIA front end to interact with the legacy system and (ii) a server-side connection layer to allow this interaction. The modernization process outlined before comprises a series of complex challenges so we try to provide the engineer with a systematic method and a partially automated toolkit.

In this paper we focus on the extraction of the navigational information from the legacy system artifacts (source code, pages, configuration files, etc.). First, we identify and locate the navigational information scattered over the WA artifacts. Then, we provide the engineer with the tool and method to generate model representations of such information. And finally we detailed the final representation we have devised to specify the navigation flows of the WA in an integrated and homogeneous way. The Struts web framework is clearly widespread through the web industry. So there exists a wide range of web applications, some of them publicly available, that may come to an interesting source of case studies. That is the rationale behind the selection of such framework for this work..

The rest of the paper is structured as follows. Section 2 presents an overview of the MIGRARIA project. In Section 3 an illustrative example is depicted. Section 4 introduces our approach to extract navigational models from a Struts-based legacy WA. In Section 5 we provide the results obtained in a study case for validation. The related work is discussed in Section 6. Finally, main conclusions and future work are outlined in Section 7.

2 MIGRARIA Project Overview

In this section, a general overview of the MIGRARIA project is introduced to provide the proper context to the work presented in this paper. One of the leading ideas of this project is to use model driven techniques and tools to deal with the complexity of extraction and interpretation processes [12]. As figure 1 presents, our approach is fundamentally organized in two different stages: (1)generating

[1] `http://struts.apache.org/`

of a conceptual representation of the legacy WA; and (2). generating the RIA front-end and its infrastructure . Three information extraction plans have been defined, marked as 1-A, 1-B and 2-B, and one restructuring process, 2-A. Our purpose is to obtain information from three complementary views of the same system: (1) the dynamic runtime view at client-side; (2) the static view of the system source code at server-side; (3) and the dynamic view of user interaction trace from server-side runtime log.

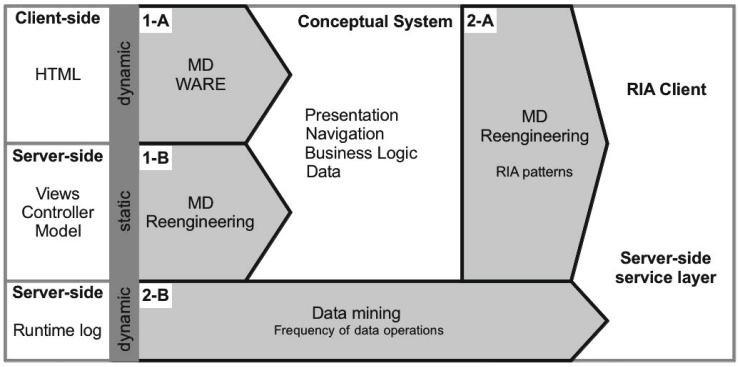

Fig. 1. MIGRARIA overview

Regarding action plan 1-A, we apply Web Application Reverse Engineering (WARE) techniques [9] to perform dynamic analysis mimicking the user interaction with the WA. And representations of different concerns of a WA are generated, such as its navigation map or its inferred data model.

Concerning action plan 1-B, we apply model driven reengineering methods to perform static analysis of all the different sources available at server side , such as: the source code of the views and the controllers, database schemas, configuration files. Again our purpose is to generate a conceptual representation of the legacy system by producing models of its conceptual layers.

Then, approaches 1-A and 1-B share a common objective but they follow different strategies. Our aim is to get two complementary conceptual representations of the same legacy WA. Both of them will be mixed, in a later stage, to get a more precise representation, decreasing ambiguity and knowledge loss derived from information extraction and interpretation activities.

On the other side, action plan 2-B consists of the application of data mining techniques to extract the user interaction trace from the server log files. In this case, we use the statistical information retrieved to derive a proposal of the RIA Client UI composition, because a compelling RIA Client UI composition cannot be derived right from a mere mapping of the navigation and presentation layers of the WA obtained at the fist stage.

Finally, the second stage of our approach is mainly realized by action plan 2-A. We apply model driven techniques to restructure and to evolve the conceptual

models of the WA into conceptual models of its RIA counterpart. In [13] we introduced an approach based on model weaving and RIA patterns.

3 Illustrative Example

The Agenda[2] system is one of the case studies used within the MIGRARIA project. Agenda is an example of data-driven WA since the web layer of the system mainly consists of a CRUD client that interacts with the underlying information system. Several frameworks and Java stack technologies for WAs have been used in the development of the system, as Struts for the web layer.

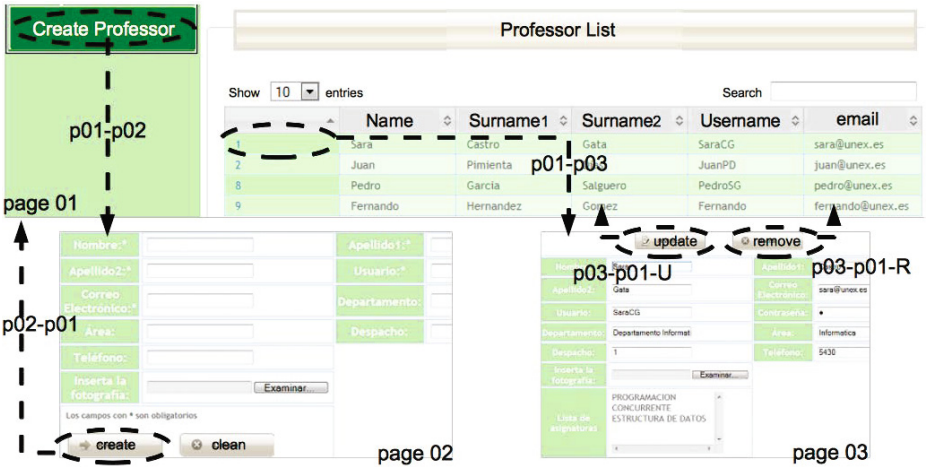

Fig. 2. Pages and flows of the illustrative example

Figure 2 shows the professor management process and includes all the CRUD operations related to the professor data entity resulting in different navigational flows departing and arriving to *page01*. This part of the system is representative enough to be an example of the most common navigational flows used in this system. Observe in the figure that the page containing the list of professors (Display action) is marked as *page01*, the professor sign in page (Creation action) is marked as *page02* and the removing and updating page (Remove and Update actions) as *page03*. We have also identified the navigational flows between these pages in order to be referenced in subsequent sections. The example contains 5 different navigational flows identified as: *p01-p02* flow (*page01 Create Professor* link), *p02-p01* flow *(Page02 create* button), *p01-p03* flow (*page01* list item links), *p03-p01-U* flow (*page03* update button), and *p03-p01-R* flow (*page03* remove button).

[2] `http://www.unex.es/eweb/migraria/cs/agenda`

4 The Approach

The main goal of this work consists of the extraction of navigational models from WAs developed with MVC-based web frameworks. As shown in section 2, for the first stage of our approach, information is extracted from a legacy WA by two different means: (1) applying WARE techniques, and (2) reengineering the original WA. In this work we focus on the latter, so figure 3 describes the main steps of our model driven reengineering process (labelled 1-B at figure 1). As previously shown, for practical purposes, Struts 1.X[3] has been selected as the reference web framework for this work.

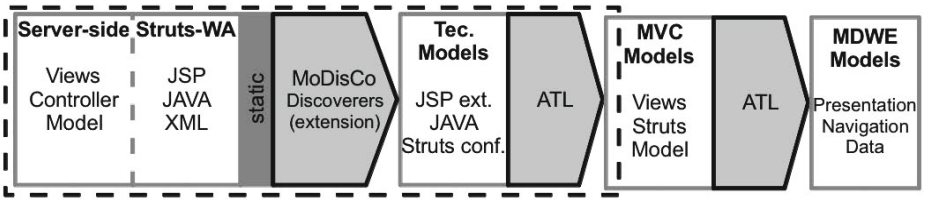

Fig. 3. Model Driven Reengineering

As input, our process takes the source code of a Struts-based WA (JSP, Java, XML) to perform a static analysis. First of all, we use MoDisCo [2] discoverers to generate models directly from the source code (text-to-model transformation). MoDisCo gives us a model representation of every source code resource of the WA (every JSP, Java, or XML file). However, MoDisco discoverer outputs not Struts-aware models, i.e Struts main concepts are not first level entities. So those models are complete but too complex to derive navigational models from them. In this work we have adapted MoDisco to create Struts-aware models from our legacy application. We then use these models to produce a representation of the WA on a higher level of abstraction conformed to our MVC (Struts) metamodel. This transformation is specified by the definition of ATL Rules. On this stage, our main concern is to get an accurate specification of the navigational aspect of the WA. With that purpose, we build a MVC (Struts) model that collects all the interaction flows (and the elements involved) defined on the web layer, generating a comprehensive view of the navigational concern.

Following, the main elements and activities involved in the first section of our process are detailed, surrounded by a dashed line at figure 3.

4.1 Locating Navigation Information

In this work, we are only interested in the information related to the navigational concern of the WA (page linking and data transferring). In MVC web

[3] `http://struts.apache.org/1.x/`

frameworks, the navigational information is scattered throughout the views and the controller specification so that the encapsulation of this information into a software artifact (model) is also a contribution of our work. The Struts architecture is derived from a combination of the Front Controller and Intercepting Filter patterns. Struts provides a single controller that governs the application events, while filters catch and process incoming and outgoing events to ensure that each of the MVC components receive exactly the information they need. The framework also provides custom tags for communication between these layers.

To define the views, Struts provides four different taglibs created by applying the JSP extension mechanism[4]: bean, html, logic and nested. Table 1 shows a summary of the main tags that provide information regarding the navigational concern,useful for our extraction process. Basically, we are interested in those tags that define server requests and their parameters. In HTML, these requests are mainly represented by form submissions and requests performed by means of hyperlinks (*anchor* tag). Struts defines its own JSP tags to generate dynamically this type of HTML elements. These tags are defined within the HTML taglib and they generate HTML 4.01 or XHTML compliant outputs in Struts.

Table 1. Information extraction summary

Taglib	Tag	Relevant information
html	form	Requested Action (ActionMapping)
		Optional attributes for form bean
	common form tags	Request parameters (name)
		Name attribute: form bean name
		Property att.: field name and bean property name
	link	4 different types: forward, href, action and page
		1 request parameter, attributes: paramID, paramName, paramProperty, paramScope
		Multiple request parameters, attributes on a Map: name, property and scope
logic	redirect	3 different types: forward, href and page
		Attributes for parameter specification as html:link
	forward	Attribute name: global ActionForward

Regarding the Controller component (MVC pattern), Struts provides three main components to define it: (1) ActionForm classes to manage and encapsulate data serialization and validation; (2) Action classes to handle each logical request that may be received by the WA; and (3) ActionMappings to relate each logical request to its corresponding handler. The two formers are defined in JAVA whilst the latter is defined by using XML (conformed to the struts-config DTD). In Struts 1.X, the ActionsMappings database is a key element for defining navigation in the system. Each ActionMapping allows the developer to relate a set of requests with the action (or actions) that handles them, including

[4] `http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/JSPTags.html`

all the information needed to deal with these requests (ActionForms as request parameters, forwards as replies to the request, etc). On the other hand, Actions contain the code that: (1) populates the data for the views; (2) handles the operations to modify the model; and (3) directs the control flow (navigation), providing a particular view as reply to the request or passing the control to a different action (forwards).

Regarding the model, Struts does not explicitly provide any related element. For the system state maintenance, a Data Access Framework is usually used in order to keep the code independent of the persistence actions: separate the business logic from the role that Action classes play.

In order to make the information extraction process systematic, we have defined a set of base navigation flows that categorize the ways that Struts deals with requests. These cases will drive the definition of the information extraction queries. Table 2 presents the different base types we have observed in our case study.

Table 2. Application base flow cases

Base case	Request	Parameters			Forwards	
		0	1	N	Page	Action
Link to a page	html:link				X	
Link to an action	html:link	X	X	X	X	X
	logic:redirect					
Form submit	html:form			X	X	X

As may be observed in Table 2, each base case is characterized by the tuple (request, parameters, forward), so we actually get 9 different cases. In a Struts-based WA, these cases does not usually appears isolated, instead the same action often handles different requests that may generate different navigation flows. Thus, we may also find two main types of actions (situations): the ones that define only one navigation flow and those that define multiple flows. It is worth noting we only consider navigation flows derived from different request (i.e., with different request parameters). We are not considering exceptional situations, such as error navigation flows, because we want to extract regular navigation flows and to avoid unnecessary complexity.

Figure 4 shows a whole example of a navigation flow specification in Struts. As may be observed, the view specifies the details of a request by using the *html:link* tag (request tuple (link to action, 1 parameter, JSP)). The mapping between this request and the action handler is specified in the configuration file by defining an ActionMapping. Finally, the execute method of the action handles the request and provides a reply. In this case, based on the request parameters, the action will reply an OK forward and the view defined in *professordetail.jsp* which presents the details of the selected professor. As the example denotes, it is common practice to write Actions that both navigate to a page and handle forms submitted from that page. Its general form is to hard code the mapping decision,

depending on the value of a request parameter, inside the execute method of the Action and to use a single ActionMapping in *struts-config.xml* to configure it [6].

JSP

```
<logic:iterate id="professor" name="professorList">
<tr class="gradeA">
    <td>
        <html:link action="professorDetail" paramId="id" paramName="professor" paramProperty="id">
            <bean:write name="professor" property="id"/>
        </html:link>
```

config

```
<action path="/professordetail"
        type="com.university.professor.ProfessorDetail"
        name="professorForm"
        scope="request">
        <forward name="ok" path="/WEB-INF/jsp/professor/professordetail.jsp" />
        <forward name="list" path="/professorlist.do" />
</action>
```

action

```
if((af.getUpdate()==null)&&(id!=null))
        forward="ok";
else{
     if(af.getUpdate()!=null)
        forward="list";
}
```

Fig. 4. Navigation flow specification in Struts

Navigation Paths in the Case Study. Regarding our running example, Table 3 shows the information to be extracted from the ActionMappings and their relationships with the navigational flows described in Section 3. As the example denotes, two of the three ActionMappings considered follow the anti-pattern aforementioned: a single action both navigates to a page and handles forms submitted from that page. Both, createProfessor and *ProfessorDetail* respond in a different way to the same request with different parameters. On the other hand, if the request does not contain data (page01 as source) they forward to page02 and page03 respectively, whilst forward and returns to page01 if the form contains data, processing previously the operation with the data contained in the form. ActionMapping *ProfessorDetail* may be considered a special type of this pattern: one action responds to three different requests (multiple submit handling). The first ActionMapping is related with the request of the action that generates page01 (the source of this request is out of the scope of the example considered).

4.2 From Struts Code to Struts-Aware Models

On one hand, a Struts-based WA is basically conformed by JavaServer pages (HTML extended with Struts taglibs, XML), the FrontController configuration

Table 3. Navigation information in the ActionMapping instances considered

Page	Page Name	Request	Parameters	Action	Forward	Nav. Path
		link to action	No	ProfessorList	Page: P01	
01	ProfessorList	link to action	No	CreateProfessor	Page: P02	p01-p02
	ProfessorList	link to action	Request: OID	ProfessorDetail	Page: P03	p01-p03
02	CreateProfessor	form	form bean: professorForm (submit: create)	CreateProfessor	Action: listProfessor (P01)	p02-p01
03	ProfessorDetail	form	form bean: professorForm (submit: update)	ProfessorDetail	Action: listProfessor (P01)	p03-p01U
	ProfessorDetail	form	form bean: professorForm (submit: delete)	ProfessorDetail	Action: listProfessor (P01)	p03-p01R

file (XML), Actions and ActionForms (Java code). So there are three different kinds of information sources: (X)HTML, XML (DTD Struts config) and java code.

On the other hand, MoDisco provides the modernization engineer with discoverers to extract models from common java web artifacts as JavaServer pages, XML and WA configuration files (*web.xml*). However, those discoverers present some limitations when processing WAs developed with web frameworks, because they are conceived from a technological base and not from a concrete framework point of view. That approach leads to more complex transformation scenarios due to the lost of the semantics related to the web framework. So we propose an extension to MoDisco to support the extraction of models closer to Struts concepts. Such extension may be approached by two different means: (1) defining new discoverers; (2) defining transformations to refine the output of MoDisco regular discoverers.

Concerning the MoDisco discoverer for JSP, it defines a JSP metamodel as an extension of the XML metamodel that covers the concepts defined in the JSTL (JSP Standard Action Eclass). Any other tag defined by the JSP custom taglib support, as Struts taglibs, is extracted as generic JSP Action elements losing its structure and semantics. In order to get a more precise representation of Struts-specific tags we have defined a new metamodel to define every Struts tag as an extension of MoDisco JSP metamodel. Figure 5 shows an excerpt of our Struts taglibs metamodel (the root EClass is JSP Action at JSP metamodel, not shown). We also provide the corresponding transformation rules to refine the JSP models obtained with MoDisco into our Struts taglib models. So, in this case, the second MoDisco extension approach has been followed.

To extract the model representing the information of the Struts configuration file we have followed the first extension approach previously proposed: the definition of a new discoverer. The development of this discoverer is based on the Web Application discoverer that processes *web.xml* files. Basically, it is conformed by two main steps: metamodel generation and model extraction. In this case, by

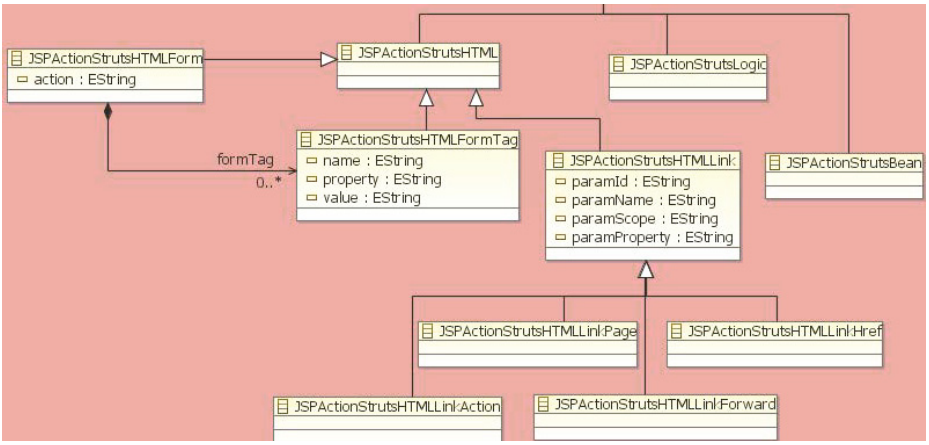

Fig. 5. Struts taglib metamodel excerpt

means of Eclipse EMF functionality the Struts configuration metamodel (Struts-conf, for short) is automatically generated from the XML Schema Definition of the Struts configuration file. So we may easily generate a Struts-conf metamodel for every version of this schema (actually 4 different versions for Struts 1.X). Once the Struts-conf metamodel is available, the extraction process is straightforward because EMF functionality permits to serialize a XML file to a XMI file conforming to the resulting metamodel, i.e. a model representing the original XML file.

Finally, regarding the java code artifacts, no extension is currently proposed because its suitability (or necessity) is under study right now. So, in this case, our process consumes the model artifacts produced by the Java MoDisco discoverer.

4.3 Modeling Navigational Information

To provide a comprehensive view of the navigational information extracted from the legacy system, an Ecore metamodel has been defined, named the Struts metamodel[5] (not detailed in this work). This metamodel allows specifying the elements of the Struts framework but also their relationships in order to define the different navigational flows of the legacy WA. As aforementioned, in a Struts-based WA, the ActionMapping database plays a key role to relate all the elements involved in a navigation flow. So we have tried to maintain Struts semantics by considering ActionMapping as the main concept of the metamodel. That way we try to simplify the automatic generation process of models conformed to our Struts metamodel, collecting just useful information (navigational concern, in this work) for its later processing.

[5] `http://www.unex.es/eweb/migraria/cs/agenda`

Generation Process. Following, a brief description of the generation process of the Struts model from the models obtained in the first step is depicted. This generation is implemented as a model transformation by means of ATL rules. The process follows the next sequence to treat the different resources: configuration file, JavaServer pages and java code (Actions and ActionForms).

The entry point of the transformation process is the rule that generates the root element of the target model: a Struts EClass instance. Then, all the global concepts (form beans and global forwards) are properly processed and their counterparts (data containers and forward instances) are created in the target model. And, to finish with the configuration file, every ActionMapping is treated: (1) resolving its references with the global elements; (2) generating the target controller Action instance; and (3) generating the local forwards instances. It is worth noting our Struts metamodel discriminates between ActionForward and PageForward elements according to the forward target type. So the rule processing forwards should be aware of this discrimination and proceeds correspondingly, i.e. selecting the appropriate EClass and filling its references properly.

Once the configuration file has been processed, we query the JSP model to extract the information of the requests contained in every page (*html:form* and *html:link* instances). Then, every request instance of the target model is related to the ActionMapping instance according to its *path* attribute. And the request parameters are specified by means of parameters instances.

Finally, the Java code is processed. Actually, the *Action* processing rule is invoked from the *ActionMapping* rule, so the different portions of the java code model are processed on demand. The relevant information to extract from an *Action* is the correct identification of the navigation flows that treats. In concrete, we are interested on identifying the input (request parameter set conforming the conditional expression granting access to a concrete control flow) and output (returned forward) of every navigational flow. That way, the corresponding references between request, action and forward instances can be established in the target model.

Struts Model of the Illustrative Example. Taking as a base our Struts metamodel, figure 6 shows the final model obtained for the driving example (*p01-p02* flow). This way, it can be seen in a single view all the elements of the model belonging to the definition of a navigation flow. For example, in the *p01-p02* navigation flow, we have:

- Origin: the page *page01* and the request *request01* contained in this page.
- Mapping: the *ActionMapping AM_RegisterProfessor* which is related to the action *CreateProfessor* by means of the action attribute.
- Target (response): the action *CreateProfessor* for the request *request01* follows the forward *PageForward01* that returns the page *page02* as response.

The flow *p01-p02* has not an explicit parameter passing between both pages. So, the entity *request01* is not related with any *RequestParameters* or *Form* instances, that act as data containers in the metamodel. The opposite situation is represented by the flow *p02-p01* where there is a data transferring (represented

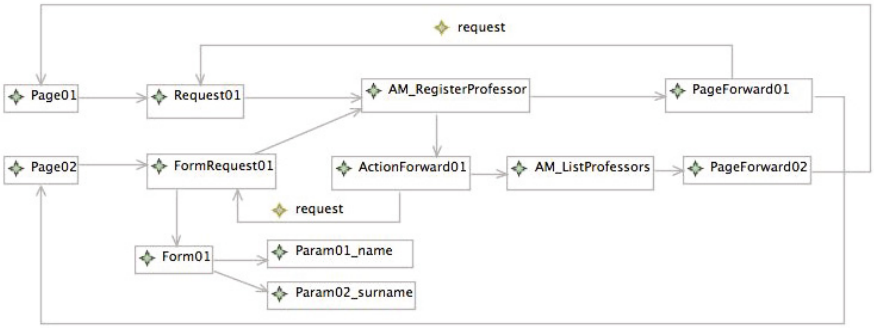

Fig. 6. Struts Model

by the entities *Form01* and *Param0X_X*) related to the operation of creating a new register in the database. Moreover, *p02-p01* flow involves an action chain: *AM_RegisterProfessor* action forwards to *AM_ListProfessors* action that finally forwards to *page01*.

It can be observed how all the navigation flows are represented in a homogeneous way, easing their querying and extraction. Moreover, all the elements implied in a navigation flow are reflected in a single model. This way, we can obtain a representation of the navigation layer of the legacy WA following one of the existing MDWE approaches (not shown in this work). Additionally, *Page* and *Action* instances include a reference to its corresponding element instances on the Struts-aware models generated in the previous step. So direct access is granted to relevant information for other concerns, such as page composition structure.

5 Evaluation

In order to evaluate our approach, we have followed a simple comparison strategy between two different versions of the Struts models for our case study: (1) one built manually by an engineer; and (2) one generated automatically by our process.

Regarding comparison results, table 4 presents some of the main results we have obtained. As shown, our approach generates automatically more than the

Table 4. Evaluation results

	Pages	Link Reqs	Form Reqs	Action Flows	Action Forwards	Page Forward
Manual	50	36	13	99	23	76
Automatic	50	30	13	119	29	90
total diff	0	-6	0	+20	+6	+14
% diff	0%	-16,67%	0	+20,20%	+26,09%	+18,42%

97% of the expected elements related to navigational information, although it underestimates link-based requests and overestimates action and forward elements. For the shake of brevity we only focus on the results indicating limitations of our approach:

- Only link requests with 0 or 1 parameter are considered (h:link-0 and h:link-1 columns). Struts use map beans to specify links with multiple parameters (h:link-n column). Actual decoding of these maps is not covered in this work.
- We only deal with link requests specified by means of html:link tags. Additionally, request parameters specified as a query string within an URL are not currently parsed. Just the action is conveniently identified for action mapping.
- Forward instances differentiation is evaluated by the equality of the values of their path attributes. Some forward instances with different path values may be considered as duplicates because the actual forwarded pages are clones located at different paths.

6 Related Work

Web Application information extraction has been performed by reverse engineering techniques [9]. Although those approaches obtain similar results to those presented herein, we consider they follow the alternative strategy identified as 1-A plan in MIGRARIA overview.

Although our intention is to follow the guidelines proposed by Architecture Driven Modernization (ADM) [16] , we have declined to use Knowledge Discovery Metamodel (KDM) because of its complexity and lack of a comprehensive definition for user interface representation. In that sense, MoDisco [2] is a generic, extensible and open source approach for software modernization that makes an intensive use of MDD principles and techniques which could be used as base to implement ADM. Our work presents a specialization of the framework defined by MoDisco to be applied in concrete modernization scenarios from Struts-based legacy WAs into RIAs.

Framework-Specific Modeling Language (FSML) [1] is a DSL to support the development of framework-based applications. The framework models are expressed using FSMLs that capture the framework abstractions as language concepts. FSML has been applied successfully to migrate a WA from Struts to JavaServer Faces. But it is a migration proposal defined at a low-level of abstraction and, though interesting, meanwhile we are interested on generating a conceptual representation of the navigational concern of a legacy WA in order to propose a modernization approach. Moreover, the migration of the views (JSPs) is not considered in that approach.

In [3] and [4] the authors introduce a process to extract models from Struts systems. In [4], the authors use two different DSLs to generate the models from the system based on source code conformed to a grammar and not well-formed source code, respectively. Based on these models, they define several model transformations that generate a JavaServer Faces version of the system. In [3] the

authors complete the work by adding an intermediate layer where KDM is used to represent the models of the system. Note that the goals of this work and ours are slightly different. The former proposes a migration of a legacy system based on a web framework to a system based on a different framework whilst our work presents a modernization approach to a RIA. The Struts metamodel presented in [4] covers the representation of the configuration file and the code related to the actions. This metamodel is on a level with the models we generate by means of our extended version of MoDisco. In our case, we propose a discoverer able to treat any version of the Struts 1.X configuration file and define a new metamodel to represent the Struts taglibs (JSP definition). Moreover we define a conceptual Struts metamodel that yields on a higher level of abstraction and provide us with a integrated view of the navigation concern of the legacy WA.

7 Conclusions and Future Work

This work outlines MIGRARIA project, an approach for systematic WA-to-RIA model driven modernization, whose main goal is to generate a RIA client and its infrastructure. In this paper, we have specially focused on extracting navigational models from Struts-based Web Applications.

By means of a running example (excerpt of a case study) we have detailed the main activities (locate, represent, transform) and artifacts (code, metamodel, model, transformation rules) related to the extraction process. The process is driven by different model artifacts that allows to define a systematic and reusable process. We have specified our own set of Struts metamodel (Struts taglibs, Struts-config and Struts metamodel) to define intermediate navigation models that remain independent of any MDWE approach. Those intermediate models may be projected to the selected MDWE approach by means of model transformations. So the methods and tools of a concrete MDWE approach may leverage our modernization process.

As main lines for future work on navigation extraction we consider the following: (1) eliminating most of the current limitations of the approach; (2) refining and validating the approach with a larger set of case studies; (3) extending the approach to support uniformly a set of MVC-based web frameworks; (4) complementing the approach with WARE strategies; and (5) defining a comprehensive tool chain to assist the whole extraction process.

References

1. Antkiewicz, M., Czarnecki, K.: Framework-Specific Modeling Languages; Examples and Algorithms. Technical Report 2007, Electrical & Computer Engineering. University of Waterloo, Waterloo (2007)
2. 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)

3. Cánovas Izquierdo, J.L., Molina, J.G.: A Domain Specific Language for Extracting Models in Software Modernization. In: Paige, R.F., Hartman, A., Rensink, A. (eds.) ECMDA-FA 2009. LNCS, vol. 5562, pp. 82–97. Springer, Heidelberg (2009)
4. Cánovas Izquierdo, J.L., Sánchez-Ramón, Ó., Sánchez-Cuadrado, J., García-Molina, J.: DSLs para la extracción de modelos en modernización. Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos 2, 1–10 (2008)
5. Di Lucca, A.G., Distante, D., Bernardi, M.L.: Recovering Conceptual Models from Web Applications. In: Pierce, R., Stanrney, J. (eds.) Proceedings of the 24th Annual ACM International Conference on Design of Communication, SIGDOC 2006, pp. 113–120. ACM Press (2006)
6. Dudney, B., Lehr, J.: Jakarta Pitfalls: Time-Saving Solutions for Struts, Ant, JUnit, and Cactus (Java Open Source Library). Wiley (2003)
7. 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)
8. 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)
9. Patel, R., Coenen, F., Martin, R., Archer, L.: Reverse Engineering of Web Applications: A Technical Review. Technical Report. University of Liverpool Department of Computer Science, Liverpool (July 2007)
10. 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)
11. 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)
12. Rodríguez-Echeverría, R., Conejero, J.M., Clemente, P.J., Preciado, J.C., Sánchez-Figueroa, F.: Modernization of Legacy Web Applications into Rich Internet Applications. In: Harth, A., Koch, N. (eds.) ICWE 2011. LNCS, vol. 7059, pp. 236–250. Springer, Heidelberg (2012)
13. Rodríguez-Echeverría, R., Conejero, J.M., Linaje, M., Preciado, J.C., Sánchez-Figueroa, F.: Re-engineering Legacy Web Applications into Rich Internet Applications. In: Benatallah, B., Casati, F., Kappel, G., Rossi, G. (eds.) ICWE 2010. LNCS, vol. 6189, pp. 189–203. Springer, Heidelberg (2010)
14. Rossi, G., Pastor, O., Schwabe, D., Olsina, L.: Web Engineering: Modelling and Implementing Web Applications (Human-Computer Interaction Series) (October 2007)
15. 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)
16. Ulrich, W.: Modernization Standards Roadmap, pp. 46–64 (2010)
17. 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)