IFML-Based Model-Driven Front-End Modernization

Roberto Rodríguez-Echeverría *rre@unex.es*
University of Extremadura / Quercus Software Engineering Group
Cáceres, Spain

Víctor M. Pavón *victorpavon@unex.es*
University of Extremadura / Quercus Software Engineering Group
Cáceres, Spain

Fernando Macías *fernandomacias@unex.es*
University of Extremadura / Quercus Software Engineering Group
Cáceres, Spain

José María Conejero *chemacm@unex.es*
University of Extremadura / Quercus Software Engineering Group
Cáceres, Spain

Pedro J. Clemente *pjclemente@unex.es*
University of Extremadura / Quercus Software Engineering Group
Cáceres, Spain

Fernando Sánchez-Figueroa *fernando@unex.es*
University of Extremadura / Quercus Software Engineering Group
Cáceres, Spain

Abstract

Since late 90's the use of web application frameworks has been the default choice to develop software applications inside the web domain. In parallel, Model Driven Web Engineering approaches have been defined and successfully applied to reduce the effort of web application development and reuse, fostering the independence of the implementation technology. A direct result of the success of these approaches is the elaboration of the Interaction Flow Modeling Language (IFML) as an Object Management Group (OMG) standard. However, the real fact is that there is a huge amount of legacy web systems that were developed before MDWE approaches were mainstream. The work presented herein tries to leverage IFML to modernize the front-ends of framework-based legacy web applications. In concrete, a systematic model-driven reverse engineering process to generate an IFML representation from such applications is presented.

Keywords: Web Models Transformations, Reverse Engineering, Model-Driven Web Engineering

1. Introduction

Since late 90's, widespread language-specific Web development frameworks (e.g. Struts[1]) have supported most of the actual development of Web Applications (WAs). These frameworks are

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

often strongly tied to the programming-language level, increasing the complexity of application maintenance and evolution. At the same time, within an academic context, Model Driven Web Engineering (MDWE) approaches [7] have been defined to leverage model driven engineering methods and techniques in the development of WAs increasing the independence of the implementation technology. MDWE approaches allow defining WAs in a declarative way by means of conceptual representations and provide code generation engines to tackle the constant change of implementation technologies. The recent publication of the IFML[2] standard has confirmed the convenience of MDWE approaches.

This work proposes a model-driven reverse engineering process to approximate these two different worlds of Web application development. Such process defines the necessary activities, artifacts and tools to generate a conceptual representation from a legacy web application developed by means of a Model View Controller Pattern (MVC)-based framework. In order to provide a detailed description, a concrete application scenario implemented by a concrete set of frameworks (Struts v1.3 and Hibernate v3.6), conventions (naming, configuration, etc.) and design patterns (e.g. Data Access Object pattern) has been specified. On the other hand, IFML has been selected as target conceptual representation. A case study has been performed to validate the approach, the Conference Review System (CRS)[3]. Obviously, the proposed process is described at a conceptual level and different realizations are possible.

The reverse engineering process defined is organized as a sequence of three steps: (1) technology-dependent model extraction; (2) conceptual MVC model generation; and (3) MVC to MDWE transformation. Since some preliminary works have been already published [5, 6], the work presented herein mainly focuses on steps 2) and 3) and supposes a major extension providing a comprehensive reverse engineering process not limited to the navigational concern and extending the work to generate a IFML-based specification.

The rest of the paper is structured as follows. A presentation of related work is done in Section 2. A detailed presentation of the generation of the MIGRARIA MVC model is performed in Section 3. The final transformation to IFML is introduced in Section 4. Results are commented in Section 5. And, finally, main conclusions and future work are outlined in Section 6.

2. Related Work

Web Application information extraction has been traditionally performed by reverse engineering techniques [3]. Most approaches presented in that survey propose strategies of static analysis taking just web pages as input of the extraction process. They treat the legacy system as a black box and focus on analyzing its output (HTML pages). Meanwhile, the approach presented herein proposes to extract the information concealed in the source code of the legacy system by means of static analysis. Moreover, most of these approaches are not conceived as model-driven approaches and pursue other aims that the generation of a conceptual representation.

The work presented in [2] and [1] proposes a model-driven process to generate a conceptual representation of a web application by using the Ubiquitous Web Application (UWA) approach. However, they also apply static analysis to the web application output without considering the system source code. That approach generates directly the final model without intermediate representations. Similarly, [4] proposes a reverse engineering process to obtain a WebML representation from a legacy PHP web shop application based on static code analysis. First, the source application is refactored to obtain a MVC version of it. Next, a code to model transformation into an intermediate model of the MVC web application is carried out. The last step is a model to model transformation from the the MVC model into a WebML model. However, the MVC metamodel proposed by the authors is tied to the WebML schema while the MIGRARIA-

[2] http://www.ifml.org/

[3] Additional material http://uex.be/migrariaifml

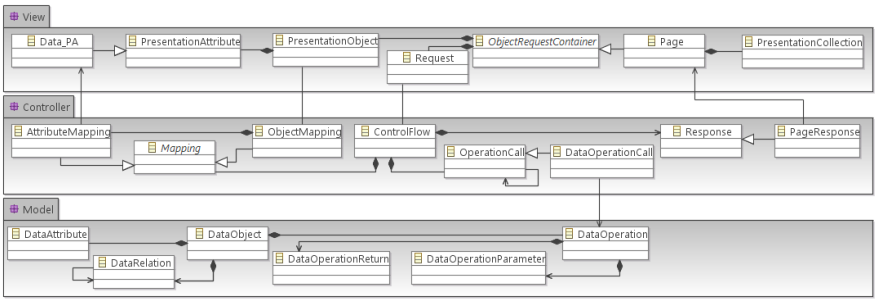


Fig. 1. MIGRARIA MVC metamodel overview

MVC one represents the general concepts defined by mainstream MVC-based web application frameworks, providing a higher degree of reutilization.

[8] presents a revision of existing approaches for reverse engineering of web applications. Most of the work related to reverse engineering of web applications has been presented before 2009. Since then the main approaches have moved to analyze Rich Internet Applications and its related technology. The authors conclude that the current trend in web application reverse engineering is the used of general reverse techniques. They also state that the use of web application frameworks make the reverse engineering process harder. In contrast we think we may take advantage of the knowledge of the framework to guide conveniently the analysis of the software artifacts.

3. MVC Model Generation

In order to generate a conceptual MVC-based representation of the legacy system is necessary to define (1) a MVC metamodel for web applications and (2) the generation process by means of model2model transformation rules.

3.1. MIGRARIA MVC Metamodel

According to the goal of the MIGRARIA project, defining a modernization process framework for legacy web applications, a specific language has been defined to generate a conceptual representation of a legacy MVC-based web application, the MIGRARIA MVC metamodel (see an excerpt in Figure 1). This metamodel specifies the main concepts of the development of a web application arranged in the three main components of the MVC pattern: Model, View and Controller. The Model package provides elements to represent data objects, their attributes, their relationships and the operations defined over them. The View package provide elements to represent pages, as main containers, and presentation objects and requests, as main containtments. Presentation objects, basically, have a set of attributes, can indicate data input or data output and can be presented individually or inside a collection. Meanwhile, requests are characterized by their parameters and path and define connection points with controller elements by means of the request-handler association. The Controller package provides elements to represent request handlers (ControlFlow), their mappings defined between presentation and data objects, their response defining a relationship with the target page element, and the sequence of operation calls performed to execute the requested action or to fetch the requested data.

3.2. MIGRARIA MVC Model Generation

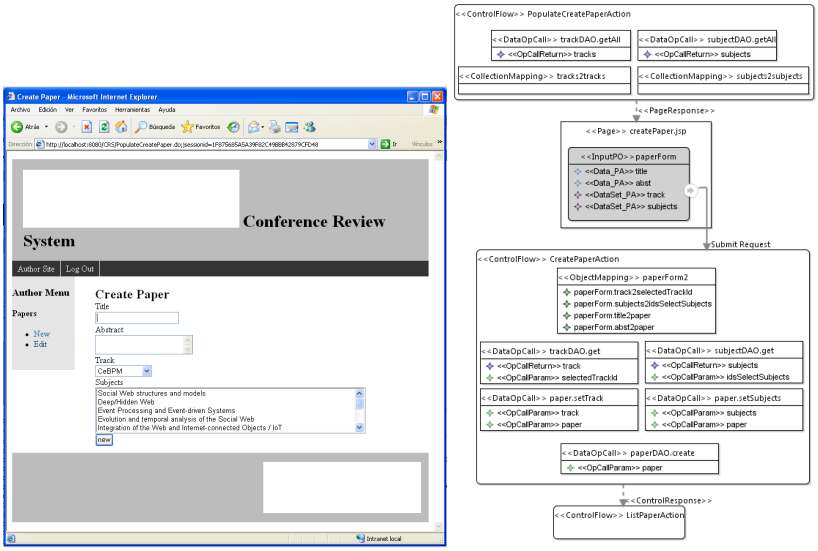
The generation process is accomplished in three sequential steps, one for each package defined by the MVC metamodel. First, all the interesting information related with data schema and data operations is collected in the model representing the Model component of the legacy system.

Second, all the information concerning web pages composition and interaction is gathered in the model representing its View component. Those two models are generated independently of each other. And finally, all the information related to request handling and operation execution is collected by the model representing its Controller component. This last model plays a fundamental role and connects the former ones by defining mappings and relationships between them.

The generation of the Model component is performed by analyzing Java data objects and data access object classes and the Hibernate mapping configuration files. Due to space reasons, it is not deeply explained.

Bearing in mind our approach is focused on data-driven web applications, next we detail how some CRUD operations are represented by our MVC model.

Figure 2 presents an example of a create operation. In this case, at the left, an excerpt of the author submission page from the CRS case study is shown that is basically composed by a HTML form allowing to specify the title, abstract, subjects and track of the paper to submit. The form contains a pair of input text controls and a pair of select controls. The JSP producing the submission page is analyzed to get its presentation objects and requests. The product of that analysis is the page *PaperCreate* of the View model. This model element is composed of an input presentation object (HTML form) that contains a pair of data presentation attributes (input text) and a pair of dataset presentation attributes (select) representing corresponding form controls. Dataset attributes have associated two presentation collections that allow to specify the set of elements populating each dataset presentation attribute. And the view also contains the specification of the request *new* that represents the submission of the form and references to the input presentation object *paperForm* by means of its *submit* attribute.



(a) Legacy Web Application **(b)** MIGRARIA MVC representation

Fig. 2. Create paper operation

As a basic common pattern, at least, two different controllers are related to a concrete view, one populating its objects and collections and the other one handling the request generated from the view. In this case, the controller *PaperPopulateCreateAction* specifies the data operations called (*trackDAO.getAll* and *subjectDAO.getAll*) and the mappings defined to fetch the data for every dataset presentation attribute of the presentation object *paperForm*. The mappings define the relation between the controller instances storing the return of the operation calls (tracks, subjects) and the corresponding presentation collections defined in the view (*tracks* and *subjects*).

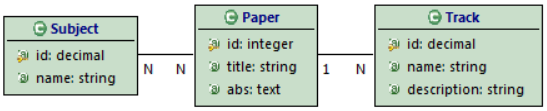


Fig. 3. WebML data model

Meanwhile, the second controller, *PaperCreateAction*, is responsible of handling the request *new*. A controller that receives a presentation object as input defines a controller instance to define the mapping. In the example, the controller instance *paper* is defined as an instance of the data object *Paper* and a mapping is established between that instance and the input presentation object *paperForm*. The mapping attributes allow defining fine grain mappings. In this case, on the one hand, the data presentation attributes of the input presentation object are mapped to the corresponding attributes of the data object paper. And on the other hand, the dataset presentation attributes are both mapped to two different controller instances: one to store the selected track id and the other one to store the ids of the selected subjects. Those instances are used as parameters of data operation calls to fetch the data objects for those ids (*trackDAO.get* and *subjectDAO.get*). The returned data objects are stored on controller instances (*tracks* and subjects) and passed again as parameters of data operation calls to associate them with the instance *paper* representing the object in creation. Finally, the last data operation call of the sequence represents the creation of a new *Paper* object from the controller instance *paper* (*paperDAO.create*).

Additionally, controllers specify its response by means of a response element. This response element may have two different types, page or control, indicating whether the controller returns directly a view or delegates the control to another handler to generate the response view. Figure 2 presents both types of response. These two different control response types allow capturing the common response patterns of a web application, such as returning a view in a HTTP GET request or redirecting to a different path (action) in a HTTP POST request. That way the association of a request contained in a page with its handler and the handler response with its corresponding page specifies the navigation map of the legacy web application.

For the sake of brevity the recovery and representation of the rest of the CRUD operations has been omitted.

4. IFML Model Transformation

In order to get a fully functional IFML specification of the legacy web application, WebRatio has been used as supporting tool in this work. This is the most featured IFML supporting tool so far. We can additionally use the professional-level model edition and code generation tools of WebRatio while keeping the specification conformed to the IFML standard. In this sense, for the sake of simplicity, regarding the data model specification, WebML Data model schema has been used. Although IFML standard does not state a concrete data representation, WebRatio requires to use WebML data representation.

4.1. Data Model

WebML Data model schema is subsumed by MIGRARIA MVC metamodel, so a simple straightforward transformation process may be defined. MIGRARIA MVC model provides an extension of WebML data schema by introducing concepts to specify the operation set related with a concrete data object. Basically, transformation rules are defined to scan all the data objects, its data attributes and relationships to generate their WebML counterparts (entities, fields and relations). Figure 3 shows an excerpt of the WebML data model generated for the CRS system concerning the entities involved in the reverse engineering example illustrating the approach.

4.2. IFML Model

In order to illustrate the details of the transformation process in a manageable way, the resulting models for the CRUD operations aforementioned are presented and the generation of every one of their elements is traced back to the MVC model.

Figure 4 presents the elements generated to model the creation of a paper (a new submission) from the MVC model of Figure 2. The main structure of this model is conformed by the following elements:

- IFML model elements:
  - A *page*, named *paperCreate.jsp*, as main container, representing the author submission page.
  - A *View Component Form*, named *paperForm*, representing the HTML form to input the submission data.
  - Two *Actions*, named *PopulateCreatePaperAction.execute* and *CreatePaperAction.execute.* The first one represents the population form operation and the second one the registration of a new paper in the system.
- *PopulateCreatePaperAction.execute* action elements
  - Two *Selector Units*, named *getAll - IN!subjectDAO.getAll* and *getAll - IN!trackDAO.getAll*, (HTML select inputs).
  - Given that action has not got input parameters, its *Input Collector Unit* is empty.
  - Its *Ok Collector Unit* contains the following four output parameters: *subjects.id[output]*, *sujects.name[label], tracks.id[output]* and *tracks.name[label]*. They are related to the 2 select controls (*track* and *subjects*) of the *paperForm* form.
- *CreatePaperAction.execute* action elements:
  - A *Create Unit*, named *CreateUnitPaper*, representing the creation of a new data object paper.
  - Its *Input Collector Unit* binds five parameters: *abst, fileLocalPath, subjects, title* y *track.*
  - A *Get Unit* to set the active user as the main author of the submitted paper.
  - An *Ok Collector Unit* which an *Ok Link* resembling the successful navigation flow.

As expected, the organization of these elements resembles the controller-view-controller pattern aforementioned. In this example, the controller responsible of populating the objects of the response view is transformed in the two selector units shown. Meanwhile the controller responsible of handling the request is modeled as action with the associated operation units presented.

5. Case Study

In order to evaluate this approach, we have developed a complete case study named the Conference Review System (CRS). CRS is a web application implemented by a developer team of our software laboratory. This system is based on the case study proposed in the First International Workshop on Web-Oriented Software Technology[4]. So partial MDWE representations

[4] http://users.dsic.upv.es/~west/iwwost01/

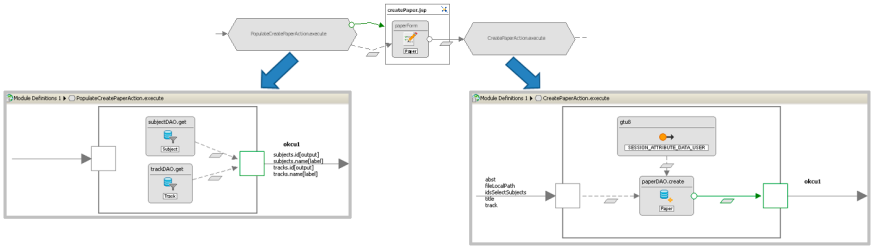


Fig. 4. IFML WebRatio create operation

Table 1. Transformation result summary

Source Element	No.	Target element	Gen.
Page	34	Page	100%
Output Presentation Objects	9	Component View Details	100%
Input Presentation Objects	16	Component View Form	100%
Object Presentation Collection (mapping)	22	Component View Simple List	100%
Object Presentation Collection (DataSet_PA)	8	Componet View Simple (filter)	NA
Data Operation Call (type=create)	6	Create Operation	100%
Data Operation Call (type=remove)	4	Delete Operation	100%
Data Operation Call (type=update)	14	Update Operation	100%
Data Operation Call (type=getAll, DataSet_PA)	5	Selector Unit	100%
Session Operation Call (type=set)	1	Set Session Attribute	100%
Session Operation Call (type=get)	12	Get Session Attribute	100%
Collection Operation Call	23	Connect Unit, Disconnect Unit or Reconnect Unit	100%

are available. The implementation technology coincides with the one proposed by the application scenario used in this paper. And the development team has followed a basic collection of code conventions and design patterns.

In order to validate our approach, the reverse engineering generated IFML model has been compared to a manual IFML model representing the same original system (CRS). Table 1 presents a summary of the results obtained. In this table only the most relevant source elements of the transformation are presented. In concrete, those ones directly related to the generation of IFML model elements. 154 elements from the 1655 elements of the source MIGRARIA MVC model. 95% of the expected target elements are successfully generated. As a final step, although a proper effort assesment experiment has not been developed, the generated IFML model was used by a WebRatio experienced engineer to generate a complete specification of the legacy system. As main conclusion our approach decreased about a 70% the development time compared to model the system manually from the scratch.

6. Conclusions and Future Work

This work tackles an important part of the modernization process defined within the MIGRARIA project which faces the evolution of the presentation of a legacy Web system towards a new Web 2.0 RIA client. In this paper we have focused on the process of extracting and representing the information from the legacy system to represent it at a higher abstraction level.

The generation of the IFML model takes as input a conceptual model of the legacy web application (MIGRARIA MVC model), so its reusability depends on the reusability of the reverse engineering process defined to get this model. In this case, two main sources of change are considered: (1) a change of the code conventions and design patterns, and (2) a change of the implementation technology. For the sake of brevity, we discuss only the former one. If the implementation technology remains the same but the code conventions change, it is mandatory to adapt the model transformation chain defined. In new case studies under development (based on real applications) we have observed that most of the times the selected strategy is to add

more steps to the original transformation chain instead of adapting the rules previously defined. Those results may suggest that our transformations rules (and the code convetions considered) are generic and comprehensive enough to be applied in different modernization scenarios without heavy modification.

As main lines for future work we consider the following: (1) providing precise metrics to evaluate the initial coverage of a new case study; (2) effort assesment; and (3) improving the tool support of the reverse engineering process to simplify the engineer's decision making.

Acknowledgements

This work has been 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).

References

1. Bernardi, M.L., Di Lucca, G.A., Distante, D.: The RE-UWA Approach to Recover User Centered Conceptual Models from Web Applications. Int. J. Softw. Tools Technol. Transf. 11 (6), 485–501 (2009)
2. Di Lucca, G.A., Distante, D., Bernardi, M.L.: Recovering Conceptual Models from Web Applications. In: Proceedings of the 24th annual ACM international conference on Design of communication. pp. 113–120. ACM Press (2006)
3. Martin, R., Archer, L., Patel, R., Coenen, F.: Reverse Engineering of Web Applications: A Technical Review. University of Liverpool (2007)
4. Rieder, M.: From Legacy Web Applications to WebML Models. A Framework-based Reverse Engineering Process. Technische Universität Wien (2009)
5. Rodríguez-Echeverría, R., Conejero, J.M., Clemente, P.J., Preciado, J.C., Sánchez-Figueroa, F.: Modernization of legacy web applications into rich internet applications. Curr. Trends Web Eng. 7059/2012 (Lecture Notes in Computer Science), 236–250 (2012)
6. Rodríguez-Echeverría, R., Conejero, J.M., Clemente, P.J., Villalobos, M.D., Sánchez-Figueroa, F.: Extracting navigational models from struts-based web applications. In-Proceedings of the 12th International Conference on Web Engineering. pp. 419–426. Springer, Heidelberg (2012)
7. Rossi, G., Pastor, O., Schwabe, D., Olsina, L. eds: Web Engineering: Modelling and Implementing Web Applications (Human-Computer Interaction Series). Springer, Heidelberg (2007)
8. Tramontana, P., Amalfitano, D., Fasolino, A.R.: Reverse engineering techniques: From web applications to rich Internet applications. In: 15th IEEE International Symposium on Web Systems Evolution. pp. 83–86. IEEE, Eindhoven, The Netherlands (2013)