Improving the adaptation of web applications to different versions of software with MDA

A. M. Reina Quintero[1], J. Torres Valderrama[1], and M. Toro Bonilla[1]

Department of Languages and Computer Systems
E.T.S. Ingeniería Informática.
Avda. Reina Mercedes, s/n.
41007 Seville, Spain
{reinaqu, jtorres, mtoro}@lsi.us.es
http://www.lsi.us.es/~reinaqu

Abstract. The Model-Driven Architecture (MDA) has been proposed as a way of separating the details of an implementation platform from the problem domain. This paper shows that this approach is also good for the adaptation of software to the different versions of the same platform. As an example, Spring Web Flow (SWF), a framework that allows the definition and representation of user interface flows in web applications, has been chosen. After six months of evolution, the web flows defined with SWF 1.0 RC1 were not compatible with SWF 1.0. The paper analyzes the changes introduced by the new release, and it proposes an MDA-based approach to soften the impact of these changes.

1 Introduction

The fast evolution of technology has caused the period of time that companies take for providing new versions of their products be shortened, if they want to be up-to-date. Many times, new releases offer new and improved features, but also cause backward incompatibility. This problem is stressed in open source projects, because new versions are often released out due to their continuous interaction with end users. Therefore, it is crucial to adapt software products that are being developed with these frameworks at a minimum cost.

On the other hand, web applications are becoming more and more complex, and nowadays, they are more than just simple interconnected web pages. Thus, an important piece in its development are Web Objects [4], that is, pieces of compiled code, that provide a service to the rest of the software system. These pieces of code are often supported by open source frameworks, and as a consequence, the evolution of these frameworks has also become an important challenge in web application evolution.

The process of releasing out new versions of a framework or a software product can be considered as a software evolution process. There are several techniques for software evolution that range from formal methods to ad-hoc solutions. But the most promising ones are: reengineering, impact analysis techniques, category-theoretic techniques and automated evolution support. MDA

also seems to be a promising philosophy for dealing with software evolution, not only because it allows the separation of the domain concerns from the technological concerns, but also because design information and dependencies are explicit in models and transformations, respectively.

This paper shows how the MDA philosophy can help us to adapt easily to a new release of a framework the software artifacts produced with an earlier version. Furthermore, it analyzes the evolution process in the MDA. To demonstrate the benefits of this philosophy, the paper will be based on a case study. The case study describes how a web flow defined with Spring Web Flow 1.0-RC1 can be adapted to Spring Web Flow 1.0, a more stable release. Spring Web Flow (SWF)[1] is a component of the Spring Framework's web stack focused on the definition and execution of user interface (UI) flows within a web application. A user interface flow can be considered as part of the navigation process that a user has to deal with while interacting with a web application. For the sake of having a clear idea of the period of time between the two sreleases, it should be highlighted that SWF 1.0 RC1 was out in May 2006, while SWF 1.0 was publicly accessible in October 2006. And, although both versions share most of the concepts, there are some technical details that differ and that cause backward incompatibility.

The rest of the paper is structured as follows: In section 2, the problem is introduced by example, that is, the working example is explained and problems are briefly highlighted. Secondly, the approach is explained following three stages: metamodel and tranformation definitions, evolution analysis and change propagation. After that, some related works are enumerated and, at last, the paper is concluded and some future lines of work are pointed out.

2 Problem Statement

Due to the constant evolution of technology, new versions of software products are released out more and more frequently. This cycle of new versions is specially speeded up when dealing with open source products. This is due to user participation: users are constantly sending reports about mistakes. In this context, a frequent operation is software migration, thus the study case is going to be focused on a migration from Spring Webflow 1.0 RC1 to Spring Webflow 1.0, a new, more stable release, which appeared just 6 months after the public appearance of the 1.0 RC1 release.

A *flow* defines a user dialog that responds to user events to drive the execution of application code to complete a business goal. And, although the elements or constructs needed to define a web flow are the same in both versions, the technical details differ from one release to the other. As a consequence, there is no backward compatibility.

In order to be clear enough, the flow used as study case is simple, but it covers the main issues needed. The initial example has been obtain from [2],

[1] The Spring Web Flow Home Page: `http://opensource.atlassian.com/confluence/spring/display/WEBFLOW/Home`

and can be seen as part of the navigation path from a simple e-shop web site. The flow simulates part of the dialog that takes place when a user wants to buy certain product. The navigation steps that a user has to pass through to buy some thing are: Firstly, select the `Start` link. Secondly, enter his personal data. Thirdly, select the properties of the product. And, finally, after pushing the `Buy` button, he will obtain a message reporting about the shopping.

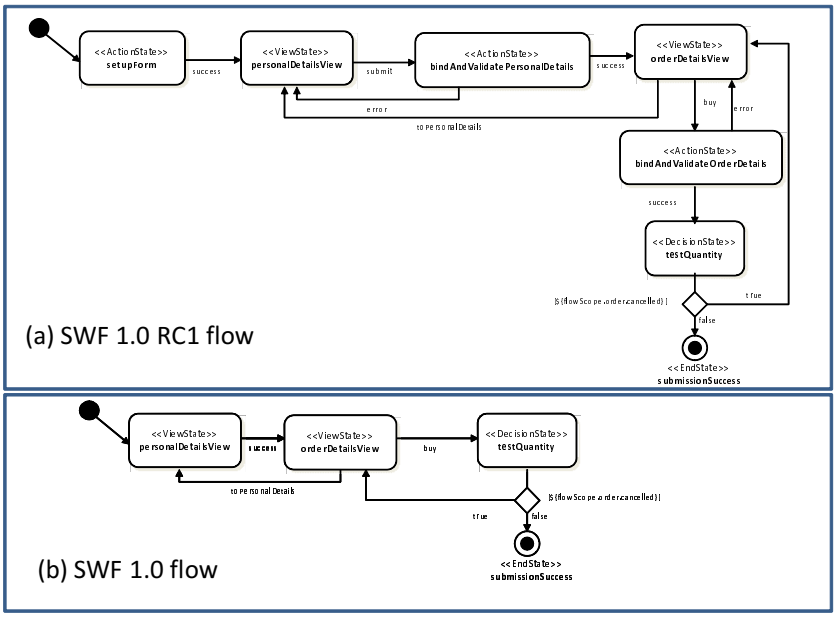

Fig. 1. State chart corresponding to the web flow specified in SWF 1.0 RC1

The implementation of this dialog using SWF requires the definition of a web flow. This web flow can be specified in two different ways, by means of an XML-file or using a Java-code. It is a good technique to draw a state chart in order to understand the web flow better. The flow consists of six states (Fig. 1(a)): three `ActionState`'s, two `ViewState`'s and one `DecisionState`. Initially, the flow starts its execution by an `ActionState` which is in charge of setting up the form. Then, the form is displayed through the `personalDetailsView` state. After that, the flow enters into another `ActionState`, which will bind and validate the data introduced by the user. If there are any problems with data, the flow will go back again to the `personalDetailsView` state; otherwise, it will enter into the `orderDetailsView` state, and the process will be repeated. Finally, if all data are right, the flow will enter into the `testQuantity` state, a `DecisionState`, that can route the flow depending on the value of the attribute `cancelled`. However, if the flow evolves in order to be SWF 1.0 compliant, we have the state chart

shown in Figure 1(b). The number of states has been reduced from six to three. That is, all the `ActionState`'s have disappeared.

3 The approach

3.1 Metamodel and transformation definition

The first step in the approach is to obtain a metamodel which expresses the concerns that are implicit in the framework and their relationships. In our case study, a metamodel of Spring Web Flow is needed. In this case, two different metamodels should be defined, one for the SWF 1.0 RC1, and the other one for SWF 1.0. The great advantage here is that, at this point, both metamodels should not differ too much. The models conforming to these metamodels are also needed. It is likely that we have models conforming to SWF 1.0 RC1, but if the model-driven process has been not followed, some reengineering techniques could be applied to obtain them. Finally, a set of transformations for obtaining the code should be given, whether model-to-text or model-to-model transformations. Furthermore, with all these artifacts, an analysis should be done in order to determine the kind of evolution that has been entered into the new release of the framework.

3.2 Analysing the evolution

In order to face up to the adaptation process, the artifacts that are subject to change should be identified, and also, which of these changes should be classified as evolution. In MDA there are three ways of evolution: model evolution, transformation evolution and metamodel evolution. In model evolution, changes to source models must be mapped into changes to target models through the transformation rules. In transformation evolution, changes to the transformation definition must be mapped into changes to target models. Finally, in metamodel evolution, changes to a metamodel must be mapped to changes to described models, plus to transformation definitions.

This section will analyze the different changes introduced in the Spring Web Flow framework, and it will classify them according to the ways of evolution in MDA. In our study case, the main changes introduced to Spring Web Flow are:

1. From DTD's to XMLSchemas. Although structurally, this change is important, conceptually is very simple. The only thing to do is to define a new set of model-to-text transformations. And, if the model-to-text transformer is based on templates, we only have to modify the template. This is a kind of model evolution.
2. Changing the root and the initial state. This change is also simple. While in SWF 1.0 RC 1, the root of the XML webflow was `<webflow>` in SWF 1.0, the root is `<flow>`. Moreover, the way of specifying the start state has also been modified: in SWF 1.0 RC, the initial state was specified as an

attribute of the root element `<webflow id="orderFlow" start-state="¬ setupForm">`; however, in SWF 1.0, it is defined as an XML element `<start-state idref="personalDetailsView"/>`. These modifications only imply the template redefinition. This is a kind of model evolution.

3. NEW `renderAction` PROPERTY IN THE `ViewState` This change is due to the introduction of a new property `renderAction` belonging to `ViewState`. But this new property implies a bit of conceptual change, because we should change the design of the flow in order to take advantage of the advanced features of the framework. Thus in SWF 1.0 RC1 an initial `ActionState` was needed in order to setup initially a form. If we look at the Figure 1, we will see that the start state is an `ActionState` named `setupForm`. However, in SWF 1.0 this can be represented by a property `<render-actions>` linked to the `ViewState` that is in charge of rendering the form. As a result of the new design, in the Figure 1(b), the `ActionState` has completely disappeared. This is a kind of metamodel evolution.
4. ACTIONS IN TRANSITIONS This change is not really due to the new version of SWF, but due to the inexperience of the authors with SWF when working with SWF RC1. Thus, two states (one `ViewState` and one `ActionState`) were defined in order to specify, on the one hand, a web form, and on the other hand, the binding and validation of data introduced by the user in that form. There, the `ViewState` named `personalDetailsView` is followed by the `ActionState` named `bindAndValidatePersonalDetails`, which is in charge of the binding and validation of user data. In that flow there is also another pair of states that follow the same pattern `orderDetailsView` and `bindAndValidateOrderDetailsView`. However, these two states can be replaced for just one `ViewState`, and the validation and biding can be triggered by the transition, which implies the disappearance of the `ActionState`. Thus, if we compare the Figures 1(a) and 1(a), we will see that the number of states has been reduced, and now, the `bindAndValidatePersonalDetails` and `bindAndValidateOrderDetails` have been missed. This is a kind of transformation evolution.

3.3 Change propagation

In order to migrate our application to be compliant to the new version of the framework, different actions should be undertaken. And the concrete action will depend on the kind of evolution. The easiest evolution to face up to is the model evolution. In this case, the only thing to do is reformulate the set of model to text transformations defined for generating the XML-Webflow files.

The metamodel evolution implies not only the modification of the Spring Webflow metamodel, but also the definition of new model to text transformations. In order to migrate the old Spring Webflow models into the new ones, two different strategies can be followed: one based on horizontal transformations, and the other one based on vertical transformations. Horizontal transformations [1] change the modular structure of an application at the same level of abstraction. If we think in model transformations, source models and target models should

be expressed at the same abstraction level. On the other hand, vertical transformations [1] involve the transformation of a high abstraction level model into a lower level one.

In the strategy based on horizontal transformations, besides the original model-to-text transformations, a set of horizontal model-to-model transformations has to be defined. These transformations along with the original and evolved metamodels, and the original model will be the input of a model transformation engine, which will produced the evolved model as output. Applying the model to text transformations, the evolved Spring Webflow will be obtained.

The second strategy is more aligned to MDA and it consists on the definition of a new metamodel which captures only the relevant concerns, that is, it should ignore those elements that are platform dependent. This second approach is a bit more expensive than the first one, in the sense that new artifacts are needed, and they are conceptually more complex. But it also has some advantages. Firstly, as the different versions deal with the same concepts, it is likely that the PIM metamodel will not change very often, and the important modifications should be at the PSM and transformation levels. Secondly, with the second approach we have to define a new metamodel for the new version of the product, but many times, this new metamodel is very similar to the one defined for the previous version, so this task does not suppose too much work. And, finally, if we define a PIM metamodel, we can deal with the same concepts but in other platforms. Finally the model evolution can be solved defining a set of horizontal transformations to reformulate the old models. In this case, the metamodel remains the same, and also the model to text transformations.

4 Related Work

The model-driven software evolution is a new area of interest, thus in the 11th European Conference on Software Maintenance and Reengineering a workshop on model-driven software evolution has been held. In [7] a survey of the problems raised by the evolution of model-based software systems is made and it is stated that Model-Driven Engineering requires multiple dimensions of evolution. Our approach deals with three of these dimensions. On the other hand, in [6] the drawbacks of model driven software evolution are analyzed, and as a conclusion the authors state that a dual approach is needed, that is, to use requirements evolution to generate the model specification and the test specification to validate the system. Our approach follows a top-down approach, but, at this point we are not interested in validation or verification.

In [3] incompatibilities between models and metamodels caused by metamodel revisions are faced up. The proposed approach is based on the syncronization of models with evolving metamodels, but this approach only deals with one dimension of evoulution, the metamodel evolution. [5] proposes a framework where software artifacts that can be represented as MOF-compliant models can be synchronized using model transformations, but they are focused on traceability of changes of software models . Finally, in [8], a systematic, MDA-based

process for engineering and maintaining middleware solutions is outline. In our approach, SWF can be considered as part of a corporate middleware.

5 Conclusions and Further Work

This paper has pointed out how we can take benefit of the MDA philosophy in order to have a better adaptation to the different versions of the same software product. To do so, a case study based on the Spring Web Flow framework has been introduced. An analysis of the changes introduced in the new version of the framework has been made. Furthermore, two different approaches based on model transformations have been considered to face to metamodel evolution. In this case, the following artifacts are needed: one SWF 1.0 RC1 metamodel, one SWF 1.0 metamodel, one web flow metamodel (this one, at the PIM level), two sets of model to text transformations (one for obtaining the XML file conforming to SWF 1.0 RC1, and another one for obtaining the XML conforming to SWF 1.0); and, finally, a set of model to model transformations, which will transform the web flow model (at the PIM level) into a model for SWF 1.0 RC1 and SWF 1.0, respectively.

One of the future works is the implementation, via web, of a metamodel repository, in such a way that we can count with the metamodels of the different versions of the frameworks. Thus, a set of metamodels will be publicly available in order to improve the adaptation to different releases of a framework.

References

1. K. Czanercki and U.W. Eisenecker. *Generative Programming. Methods, Tools and Applications.* Addison Wesley, 2000.
2. Steven Devijver. Spring web flow examined. *JavaLobby.*
3. B. Gruschko and D. S. Kolovos. Towards synchronizing models with evolving metamodels. In *Proc. Int. Workshop on Model-Driven Software Evolution held with the ECSMR*, 2007.
4. A. E. Hassan and R. C. Holt. Architecture recovery of web applications. In *Proceedings of the 24rd Int. Conf. on Software Engineering, 2002*, pages 349–359, 2002.
5. I. Ivkovic and K. Kontogiannis. Tracing evolution changes of software artifacts through model synchronization. In *Proc. of the 20th IEEE International Conference on Software Maintenance (ICSM'04)*, 2004.
6. H. M. Sneed. The drawbacks of model-driven software evolution. In *Proc. Int. Workshop on Model-Driven Software Evolution held with the 11th European Conference on Software Maintenance and Reengineering*, 2007.
7. A. van Deursen, E. Visser, and J. Warmer. Model-driven software evolution: A research agenda. In *Proc. Int. Ws on Model-Driven Software Evolution held with the ECSMR'07*, 2007.
8. J. P. Wadsack and J. H. Jahnke. Towards model-driven middleware maintenance. In *Proc. Int. Workshop on Generative Techniques in the context of Model-Driven Architecture*, 2002.