Generating domain specific aspect code for navigation from platform specific models in MWACSL

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

Resumen. MWACSL[1] is an Aspect-Oriented and Model-Driven approach for software development in the context of web applications. MWACSL uses domain specific aspect languages for dealing with the different aspects of a system. In this context one application is composed of a primary model and a set of aspect models, each one expressed by means of a Domain Specific Aspect Language. Furthermore, MWACSL is also based on Model-Driven Architecture as a way of separating the technological details. If we look at the horizontal dimension of MWACSL, this paper is focused on the platform specific level, while looking at the vertical dimension, the focus is on the navigational aspect. The specific platform is Spring Web Flow, a framework that has been thought for defining navigation flows. The main contributions of the paper are the definition of a metamodel for Spring Web Flow and a set of model to text transformations to generate web flows. These contributions can be considered as part of the MWACSL approach.

Palabras clave: Aspect-Oriented Software Development, Model-Driven Software Development, Domain Specific Aspect Languages, Model to Text Transformations

1 Introduction

Model-Driven Software Development (MDSD) and Aspect-Oriented Software Development (AOSD) are two new developmental paradigms that have come up in the last few years. One approach for MDSD is the OMG's Model Driven Architecture (MDA), which defines three different levels of modeling: Computation Independent Models (CIM), Platform Independent Models (PIM) and Platform Specific Models (PSMs).

Seminal works on AOSD proposed a series of Domain Specific Aspect Languages (DSALs), such as COOL and RIDL [11], for describing the different aspects of a system. However, the community opted for general-purpose languages, such as AspectJ, due to the problems that arose while dealing with DSAL's: the important effort necessary to implement a new DSAL, the difficulty for extending the weaver made ad hoc for the concrete DSAL and, the inability of combining weavers for different DSAL's. Nowadays, thanks to the contributions and tools provided by the MDSD community, many of the disadvantages of DSAL's can be softened by changing the technological space from grammarware to modelware [10].

In this context, MWACSL is an Aspect-Oriented, Model-Driven approach that proposes the definition of different DSLs for each aspect at the Platform Independent modeling level. The acronym MWACSL stands for: Model-driven, Web applications, Aspect-oriented and

[1] This work has been partially supported by the Spanish Ministery of Science and Technology: TIN2007-64119 and TIN-2007-67843-C06-03.

Concern Specific Languages (a capital letter mix-up of those letters related to the main areas that are combined in the approach). In MWACSL one application is composed of a primary model and a set of aspect models, each one expressed in a different Domain Specific Aspect Language (DSAL). We think that separation of concerns should be maintained as far as possible, but sometimes, some concrete platforms imposed by customers don't allow a clear separation of concerns. This paper introduces how to deal with a concrete aspect (navigation) at PSM and code. Spring Web Flow (SWF)[2] is the concrete platform that has been chosen because it is a framework that allows the definition and representation of user interface flows in web applications in a clear and simple way and it offers a clean separation between navigation and user interface.

The paper is structured as follows: Firstly, a general overview of MWACSL is given in order to put the reader in context. Then, in Section 3, a running example is shown. After that, the process of generating the web flow code is detailed focusing on two phases: the Spring Webflow metamodel definition(Section 4), and the model to text transformations (Section 5). Afterwards, the related work in areas such as aspect oriented modeling, domain specific aspect languages and web engineering is analyzed. Finally, the paper is concluded and some future lines of work are pointed out.

2 MWACSL

MWACSL is a model-driven, aspect-oriented approach for developing software in web environments in which aspects are defined using DSLs. Thus, one application is composed of a primary model and a set of aspect models (each one expressed in a different DSAL). AOSD gives us a horizontal separation, while MDA provides a vertical separation thanks to its different levels of modeling. At the PIM level, aspects are defined separately, each one in its own language. These aspects should be defined separately as far as possible, but this is not always viable. Let's suppose that our customer wants us to implement the application using the Java Server Faces (JSF) framework. The problem here is that JSF does not provide a clear separation between user interface and navigation. In this case, at the PIM level MWACSL will deal with two different DSALs, one for navigation and another one for user interface, while at the platform specific level, has to work with these two aspects mixed.

In this concrete paper, the focus is on the lowest levels, that is, PSM and code. Contrary to the example introduced above, we will show an aspect that is clearly separated, even in the lowest levels. This assumption has been made to focus only in the top-down generation process, without any interference from the weaving process. In MWACSL the weaving process implies the integration of several DSALs. As DSALs are defined by means of metamodels, the aspect composition model is transformed into a metamodel composition problem, that is out of the scope of this paper.

3 A running example

This section introduces an example to give a clear idea about the kind of web flows that can be defined in this platform. Figure 1 depicts a flow for a simple e-shopping[3] web application that has been adapted from [2]. A *flow* defines a user dialog that responds to user events and drives the execution of application code to complete a business goal. In the flow defined in the Figure 1, users must fill in two consecutive forms for ordering a product, one with his personal data and another one with information about the order itself. Thus, a flow with

[2] See: `http://opensource.atlassian.com/confluence/spring/display/WEBFLOW/Home`

[3] This model can be downloaded from: `http://www.lsi.us.es/~reinaqu/org.mwacsl/springWF/models/Shopping.springwf`

four states has been defined for implementing this dialog. The graphical notation used in the figure is similar to the UML notation for state diagrams. The notation is explained in more depth in [15]. The states have been stereotyped in order to highlight its type.

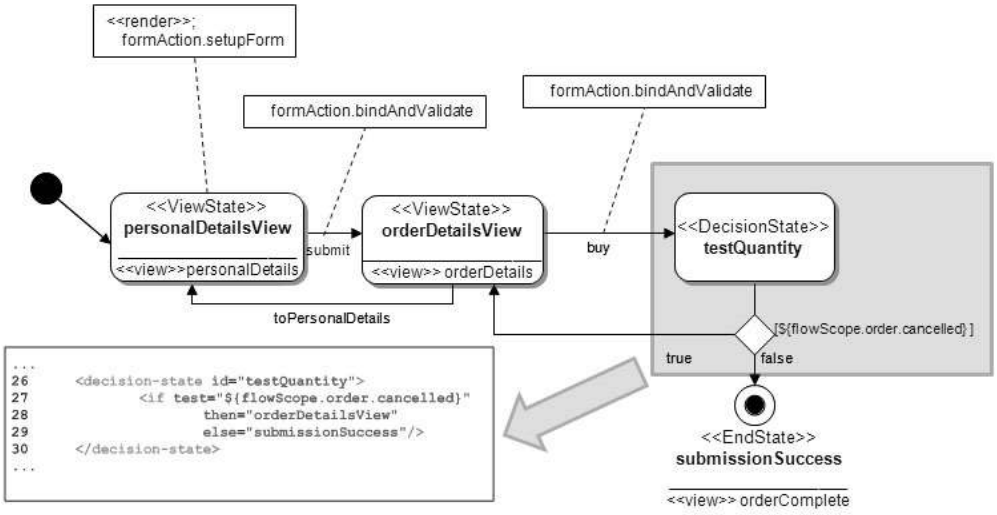

Fig. 1. A graphical Spring Webflow model for a simple e-shop application

The two first states represent two views corresponding to the two forms that users must fill in. An annotation has been link to the first `ViewState`. The `<<render>>` stereotype makes reference to the method that is in charge of setting up the personal data form. The form is specified by means of the `<<view>>` stereotype. In this way, users are asked to enter their personal data by filling in this form. When users trigger the event `Submit` by pushing a button, the entered data should be validated. In this flow, the data binding and validation is made by means of an action associated to the `submit` transition, that leads to the `orderDetailsView` state, which displays the second form. In this form users are asked for data related to the order. These data can be bound and validated with an action associated to the `buy` transition. Before ending the shopping, a test is made to check if a user has cancelled the operation. This test is executed by means of a `DecisionState`. If the operation has been cancelled, the flow will be routed to the `orderDetailsView` state and the process will be repeated. If the result of the test evaluation is false, then the flow ends its execution. The `EndState` has also associated a view to report users the shopping result.

4 From PSM to Code: The Spring Webflow Metamodel

For space reasons only a general overview of the metamodel is given in this section. An exhaustive explanation can be found in [14]. The Spring Webflow metamodel that has been defined at the PSM level for code generation purposes and it has been implemented as an Ecore metamodel[4]. The central part of the metamodel is the `Flow` metaclass, which represents a navigation flow between web pages. A flow is composed of a set of states. These states are modeled as the abstract `State` metaclass. One flow has only one initial state (where the execution starts), but it can have more than one final state. The initial state is modeled by means of the `initialState` reference which relates `Flow` and `TransitionableState`.

[4] If the reader is interested, the metamodel can be downloaded from: `http://www.lsi.us.es/~reinaqu/org.mwacsl/springWF/metamodel/SpringWF1.0.ecore`

The metamodel comprises four abstract metaclasses (`Transition`, `State`, `TransitionableState` and `Action`), whose aim is to structure the metamodel; twenty-one ordinary metaclasses; one enumeration (`ScopeType`), to cope with the scope of variables and parameters and three invariants that are linked as annotations to the `Mapping`, `Variable` and `Transition` metaclasses. Figure 2 shows these OCL constraints expressing that two attributes cannot be used at the same time, for instance, the invariant in the `Transition` context specifies that the properties `on` and `onException` can not be assigned simultaneously.

```
context Mapping
inv: (self.target = null or
          self.target.size()=0)
     xor (self.targetCollection = null or
        self.targetCollection.size()=0)
```

```
context Variable
inv: (self.class = null or
          self.class.size()=0)
     xor (self.bean = null or
          self.bean.size()=0)
```

```
context Transition
inv: (self.on = null or
          self.on.size()=0)
     xor (self.onException = null or
          self.onException.size()=0))
```

Fig. 2. OCL constraints defined in the SWF metamodel

5 From PSM to Code: The Model to Text Transformation

The model to text transformation has been implemented with MOFScript[5]. The reasons for selecting MOFScript are: firstly, MOFScript has been specifically designed for the transformation of models into text files; secondly, it can handle metamodel descriptions as Ecore files; thirdly, transformations can be directly executed from the Eclipse environment; and, finally, it provides a way of generating the output as text files.

Our MOFScript transformation is named `SpringWF2XML` and it has an input parameter that makes reference to an input model[6]. This model should conform to the Spring Webflow metamodel defined in Section 4, and it is supposed to be error free, that is, if the model has been generated by means of an editor, the editor should be in charge of doing all the model checking process. Furthermore, if the model has been generated automatically from an aspect model at the PIM level, then the model to model transformation has to produce a correct model. One example of input model is the one depicted in Figure 1 [7].

The result of executing the `SpringWF2XML` transformation is an XML file with the flow definition. The starting point for the MOFScript transformation is the `Flow` metaclass. One transformation comprises a set of rules, which are similar to functions. They can have a context (that is, the metamodel element for which the rule is defined), a return value (the context and the return value are optional), and a body that will contain a set of sentences. In general, at least one rule has been defined for each metaclass. The rule is in charge of generating the piece of XML code associated to the concrete metaclass. For example, the rule associated to the `DecisionState` metaclass is in charge of generating the piece of XML code shown in Figure 1. However, there are some metaclasses without any associated rule. This is because the piece of XML that has to be generated from the metaclass can vary depending on the path followed to reach the metaclass while navigating the metamodel. Thus, for example, there are two different composition relationships that can lead us to a `Result`: there can be a `Result` of an `EvaluateAction`, but also a `Result` can be referred to

[5] Avalaible at: `http://www.eclipse.org/gmt/mofscript/`

[6] For space reasons the transformation code has not been included in the paper, but the whole set of rules can be downloaded as an *.m2t* file from: `http://www.lsi.us.es/~reinaqu/org.mwacsl/springWF/transformations/SWF2XML.m2t`

[7] In addition to the shopping model described in Section 3, some extra models for testing purposes can be downloaded from: `http://www.lsi.us.es/~reinaqu/org.mwacsl/springWF/models/`

a method in a `BeanAction`, and the piece of generated XML is different in both cases. The solution is that the generation has been associated to the container metaclass in order to be able to evaluate the followed path.

Rules also can have any number of parameters. In our transformation, all the defined rules have a parameter named `prof`. This parameter is used for formatting purposes. It indicates the depth level of the element, which will determine the tabulation spaces.

6 Related Work

There are related works in several areas of knowledge: aspect-oriented modeling (AOM), domain specific aspect languages (DSALs) and web engineering (WE). AOM proposals can be grouped into two families: those whose goal is modeling programs that have been implemented in an aspect-oriented platform [8, 13, 4]; and those that are centered in structuring models according to the different aspects that compose them [1, 5]. Our approach is more aligned with the second family. However, as we think that current aspect-oriented languages are another implementation platform, some of the proposals that belong to the first family can be used in MWACSL at the PSM level, in order to, in a later phase, generate AspectJ or HyperJ code. Furthermore, AO approaches can be classified into asymmetric and symmetric approaches [9]. MWACSL is an asymmetric approach that it is very close to [5]. The main difference is that they use a general-purpose approach for defining aspects.

In the area of DSALs, many of the published papers are focused on the specification of a new DSAL that is specified in the grammarware technological space [10]. The most important difference with our approach is that MWACSL deals with these aspects in the modelware technological space.

Regarding to web engineering and navigation, it has to be highlighted that most of the approaches tackle the navigation design separately. In general, it can be said that there are two different views of navigation: a behavioral perspective and a structural perspective. While the proposals that design navigation from a behavioral perspective are based on the navigation operational semantics, the ones centered on a structural perspective are focused on structuring information in contexts that represent coherent pieces of information that are interconnected by means of links with meaning. The first ones are based on state machines, and the second ones on the composition and structural relationships among the navigation structures. The models introduced in this paper address the behavioral perspective because it has been imposed by the implementation platform. Some proposals that deal with the behavioral perspective of navigation are [18, 3].

Finally, there are also some proposals that are mixing AOSD, MDSD and WE [19, 16, 6, 17]. If we focused on the navigational aspect, [7] addresses the navigation routing code as an aspect, while [6, 17] deal with navigational concerns during early stages.

7 Conclusions and further work

MWACSL is an aspect-oriented, model-driven approach for the web domain, based on the modeling levels proposed in the OMG's MDA approach. MWACSL also deals with different DSALs. This paper is focused on a single aspect, navigation, at the Platform Specific Modeling level. For implementing this aspect, a metamodel for Spring Webflow (the implementation platform) has been defined. A set of model to text transformations have also been implemented, and some models for testing purposes have been created.

As further work, it has already been pointed out that we are working on defining a spring web flow editor with the Topcased modeling framework in order to generate it. We also want to address other aspects and frameworks, such as persistence and Hibernate [12], respectively. Our idea is that users can count with a high level aspect library.

Referencias

1. S. Clarke and E. Baniassad. *Aspect-Oriented Analysis and Design: The Theme Approach.* Addison-Wesley Professional, 2005.
2. S. Devijver. Spring Web Flow Examined. *JavaLobby*, 2005.
3. P. Dolog and W. Nejdl. Using uml and xmi for generating adaptive navigation sequences in web-based systems. In *Proc. of UML 2003 - Sixth International Conference on the Unified Modeling Language: Modeling Languages and Applic*, number 2863. Springer-Verlag Lecture Notes in Computer Science, October 2003.
4. J. Evermann. A meta-level specification and profile for aspectj in uml. In *AOM '07: Proceedings of the 10th international workshop on Aspect-oriented modeling*, pages 21–27, New York, NY, USA, 2007. ACM Press.
5. R. France, I. Ray, G. Georg, and S. Ghosh. Aspect-oriented approach to early design modeling. *IEE Software*, 151:173–186, June 2004.
6. S. Gordillo, G. Rossi, A. Moreira, J. Araújo, C. Vairetti, and M. Urbieta. Modeling and composing navigational concerns in web applications. requirements and design issues. In *Proc. of the Fourth Latin American Web Congress, 2006. LA-Web '06.*, pages 25–31, Oct. 2006.
7. M. Han and C. Hofmeister. Modeling and verification of adaptive navigation in web applications. In *ICWE '06: Proceedings of the 6th international conference on Web engineering*, pages 329–336, New York, NY, USA, 2006. ACM Press.
8. Y. Han, G. Kniesel, and A. B. Cremers. A meta model and modeling notation for AspectJ. In Omar Aldawud, Grady Booch, Jeff Gray, Jörg Kienzle, Dominik Stein, Mohamed Kandé, Faisal Akkawi, and Tzilla Elrad, editors, *The 5th Aspect-Oriented Modeling Workshop In Conjunction with UML 2004*, October 2004.
9. W. Harrison, H. Ossher, and P. Tarr. Asymmetrically vs. symmetrically organized paradigms for software composition. In Lodewijk Bergmans, Johan Brichau, Peri Tarr, and Erik Ernst, editors, *SPLAT: Software engineering Properties of Languages for Aspect Technologies*, March 2003.
10. I. Kurtev, J. Bézivin, and M. Aksit. Technological spaces: An initial appraisal. In *CoopIS, DOA'2002 Federated Conferences, Industrial track*, 2002.
11. C. V. Lopes. *D: A Language Framework for Distributed Programming.* PhD thesis, College of Computer Science, Northeastern University, 1997.
12. Red Hat Middleware. Hibernate web page, 2006.
13. I. Philippow, M. Riebisch, and K. Böllert. The Hyper/UML approach for feature based software design. In Omar Aldawud, Mohamed Kandé, Grady Booch, Bill Harrison, Dominik Stein, Jeff Gray, Siobhán Clarke, Aida Zakaria Santeon, Peri Tarr, and Faisal Akkawi, editors, *The 4th AOSD Modeling With UML Workshop*, October 2003.
14. A. M. Reina Quintero. Spring web flow metamodel 1.1. `http://www.lsi.us.es/~reinaqu/org.mwacsl/springWF/doc/SWF_MM_1.1.pdf`, February 2008.
15. A. M. Reina, J. Torres, and M. Toro. El metamodelado de un framework: Spring web flow. *Actas de los Talleres de las Jornadas de Ingeniera del Software y Bases de Datos (TJISBD)*, 1(3), 2007.
16. A. Schauerhuber, M. Wimmer, W. Schwinger, E. Kapsammer, and W. Retschitzegger. Aspect-oriented modeling of ubiquitous web applications: The aspectWebML approach. In *ECBS '07: Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, pages 569–576, Washington, DC, USA, 2007. IEEE Computer Society.
17. P. Valderas, V. Pelechano, G. Rossi, and S. Gordillo. From crosscutting concerns to web systems models. In *Proceedings of the Web Information Systems Engineering WISE 2007*, volume 4831, pages 573–582. Springer-Verlag Lecture Notes in Computer Science, 2007.
18. M. Winckler and P. A. Palanque. StateWebCharts: A formal description technique dedicated to navigation modelling of web applications. In J. A. Jorge, N. Jardim Nunes, and J. Falcão e Cunha, editors, *DSV-IS*, volume 2844 of *Lecture Notes in Computer Science*, pages 61–76. Springer, 2003.
19. G. Zhang, H. Baumeister, N. Koch, and A. Knapp. Aspect-oriented modeling of access control in web applications. In M. Kandé, D. Stein, O. Aldawud, T. Elrad, J. Gray, and J. Kienzle, editors, *6th International Workshop on Aspect-Oriented Modeling*, March 2005.