Model-Driven Web Form Validation with UML and OCL

Eban Escott[1], Paul Strooper[1], Paul King[2], and Ian J. Hayes[1]

[1] The University of Queensland, School of Information Technology and Electrical Engineering, Brisbane, QLD, 4072, Australia
[2] ASERT, Level 23, 127 Creek St, Brisbane, QLD, 4001, Australia
{eescott,pstroop,ianh}@itee.uq.edu.au,
paulk@asert.com

Abstract. Form validation is an integral part of a web application. Web developers must ensure that data input by the user is validated for correctness. Given the importance of form validation it must be considered as part of a model-driven solution to web development. Existing model-driven approaches typically have not addressed form validation as part of the model. In this paper, we present an approach that allows validation constraints to be captured within a model using UML and OCL. Our approach covers three common types of validation: single element, multiple element, and entity association. We provide an example to illustrate an architecture-centric approach.

Keywords: Model-driven, web engineering, web form validation.

1 Introduction

The user experience of any web application is crucial to its success. This experience is influenced by many factors, including form validation. Users will fill out forms that are submitted to the server and these forms must be validated to ensure that the data entered is acceptable. Subsequently, this data could be used for some immediate operations, such as email or to invoke a web service. It could also be stored in a database for later use. No matter the intended use of the data, it is imperative that it is validated for correctness.

Form validation can be achieved on either the client-side or server-side of the application. Client-side validation offers a richer user experience by using technologies such as JavaScript and AJAX. Solely relying on client-side validation is a risk as a user may disable JavaScript via a browser setting. Therefore, server-side validation is a necessity that should not be avoided. For this reason, and to reduce scope, we focus only on server-side validation, although our approach could be applied to client-side validation as well.

Given the importance of form validation it must be considered as part of a model-driven solution to web development. There are many proposed web-modelling languages, but most do not address form validation. In this paper, we categorise form

A. Harth and N. Koch (Eds.): ICWE 2011 Workshops, LNCS 7059, pp. 223–235, 2011.

validation and propose a model-driven solution that uses UML [1] and OCL [2]. UML is a general purpose modelling language and OCL augments UML to make more precise models. We analysed four different web application frameworks to ensure our approach can be used for a number of target platforms and show an example of generating form validation for one framework.

Section 2 discusses the related work and how other web modelling languages have included form validation. Section 3 shows how form validation is coded in a web application framework. This demonstrates the code that we must generate as part of our solution and in Section 4 we give an example of how to achieve this using UML and OCL. Section 5 discusses the results of using the approach on the example and relates the approach to generating form validation code for other web application frameworks. We conclude in Section 6 and summarise our future work.

2 Related Work

There are many different ways of applying model-driven development and each has its goals and priorities. We subscribe to Architecture-Centric Model-Driven Software Development (AC-MDSD) [3] in which the goals are development efficiency, software quality, and reusability. This is in contrast with other well-known approaches, such as the Model Driven Architecture (MDA) [4], where the goals are interoperability and software portability.

Stahl and Völter [3] describe AC-MDSD alongside an iterative two-track development process in which there is an implementation track, which is the target application, and a modelling and transformation track[1]. The implementation track is responsible for building the *reference implementation* that is used to derive the models and transformations. Stahl and Völter recommend that the implementation track should be one development iteration in front of the modelling and transformation track. The implementation track emphasises the importance of web application frameworks, which we discuss in Section 3. It may seem counter-intuitive to build a *reference implementation* as this creates an extra cost, but when considering that the outcome of AC-MDSD is to build many applications of the same software family, this cost is offset by later gains. This is not too dissimilar to a software product line where an initial investment must be made [5].

Existing web modelling languages have rarely discussed the issue of form validation. During our literature review of OOWS [6], OOHDM [7], UWE [8], IDM [9], WebML [10], Hera [11], and WSDM [12], we did not find references addressing form validation. Additionally, the code generators UWE4JSF [13] for UWE, OOHDMDA [14] for OOHDM, and HPG [15] for Hera, do not have form validation included. WebRatio [16] for WebML includes form validation by adding validation rules to entry units that are part of its domain-specific language (DSL). The tool

[1] Stahl and Völter refer to the two-track development process as having a *domain architecture* track and an *application* track. We refer to these as the *implementation* track and the *models and transformations* track respectively. We believe these terms are more intuitive in the context of this paper.

generates code for the Struts [17] web application framework but the details are not published as the tool is proprietary.

The only web modelling language openly addressing form validation is WebDSL [18]. WebDSL maintains separation of concerns while integrating its sublanguages, enabling consistency checking and reusing common language concepts. Groenewegen and Visser [19] have designed a WebDSL sublanguage for form validation and categorised form validation into *value well-formedness*, *data invariants*, *input assertions*, and *action assertions*. *Value well-formedness* checks that the input conforms to its expected type, *data invariants* are constraints in the domain model, *input assertions* are for form elements which are not directly connected to the domain model, and *action assertions* are validation checks during the execution of actions.

The WebDSL approach contrasts with our work as we apply a reusable UML-compliant solution, not a textual DSL specific to WebDSL. The WebDSL validation categories of *value well-formedness* and *action assertions* are not considered relevant in our context. For us, *value well-formedness* is handled by the web application framework and *action assertions* are not applicable to standard web form validation, which is the scope of our research. WebDSLs *input assertions* do not warrant a new category in our research as our UML profile can be applied to multiple models as discussed in Section 4.4. The last WebDSL category of *data invariants* is comparable to our validation categories. We use a finer-grained approach that is suited for specific web application frameworks.

We believe that the target web application framework is important. WebDSL attempts to create an implementation-neutral language that can be applied to multiple code generators. This approach has merit but risks missing opportunities to utilise features that exist in one web application framework and not in another. This point relates to the goals and priorities given the model-driven philosophy discussed at the beginning of this section and elaborated further in Section 5.

3 Web Application Frameworks

In our approach, the target architecture is closely aligned with a chosen web application framework. We examined four such frameworks and what the generated code might look like. This is part of an AC-MDSD approach whereby a developer should start by building the *reference implementation* first. Subsequently, this is used to abstract to the models and drive the transformations.

Web application frameworks form the backbone of modern web development. The features available for each framework vary but they all have some built-in mechanism to support form validation. For our research, we analysed four frameworks all with different programming languages. The frameworks are Spring MVC [20], Ruby on Rails [21], Grails [22], and ASP.NET MVC [23] using programming languages Java, Ruby, Groovy, and C# respectively. Spring MVC is used for our example in Section 4 based on its popularity in the market place, but we believe any of the four frameworks could be used as discussed in Section 5.

The Spring MVC web framework validation is based on JSR:303 Bean Validation [24] by the Java Community Process. JSR:303 allows developers to define declarative validation constraints based on annotations. For example, in the following program code on lines 6 and 7, a persons age has a minimum of 0 and a maximum of 110.

```
1  public class PersonForm {
2
3    @NotNull
4    private String name;
5
6    @Min(0)
7    @Max(110)
8    private int age;
9   …
10 }
```

The annotations are applicable to single HTML elements only. If the developer needs to validate multiple elements or ensure entity associations are correct then they must use a custom validator. The following program code is a custom validator for Spring. The method *validate* on line 7 is invoked and provides an opportunity for developers to add in custom validation beyond what is available via JSR:303 annotations, or implement validation without annotations as shown in the following program code.

```
1  public class PersonValidator implements Validator {
2
3    public boolean supports(Class clazz) {
4      return Person.class.equals(clazz);
5    }
6
7    public void validate(Object obj, Errors e) {
8      ValidationUtils.rejectIfEmpty(e, "name", "empty");
9      Person p = (Person) obj;
10     if(p.getAge() < 0) {
11       e.rejectValue("age", "negativevalue");
12     } else if(p.getAge() > 110) {
13       e.rejectValue("age", "too.old");
14     }
15   }
16 }
```

All of the four web application frameworks we analysed have some mechanism for standard validation and custom validation. It is important to recognise the target code generation as this will become the *reference implementation* in the iterative two-track development process described in Section 2. In AC-MDSD, it is the implementation track that drives the modelling and transformation track.

4 Example

Our example shows how we use UML and OCL for form validation. The example involves generating a web application and manually testing the form validation. The generated web application provides create, read, update, and delete (CRUD) functionality for applicable parts of the domain. Fig. 1 is the domain model for our example and it is part of a typical e-commerce web application. A product belongs to a brand, can be categorised, and purchased via a shopping cart.

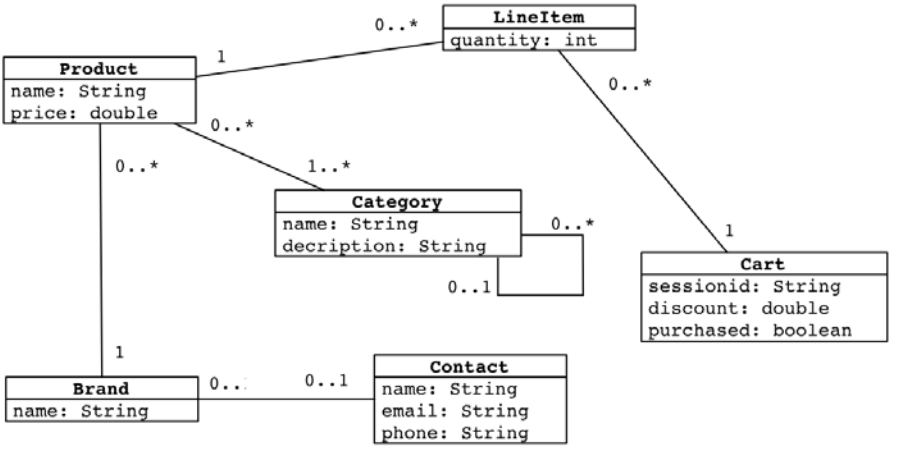

Fig. 1. Domain Model

In Section 4.1 we discuss the scope of the HTML elements we will consider for validation. In Section 4.2 we categorise the different types of validation and in Section 4.3 we describe our model-driven solution. The UML Profile is presented in the Section 4.4.

4.1 Target Elements

The W3C develops standards to ensure the long-term growth of the Web. At the time of writing this paper, the current version of HTML is 4.0.1[2]. We will be focusing our validation on these HTML elements. Section 17 of the HTML 4.01 specification [25] lists all possible form elements. Fig. 2 shows example HTML with a typical rendering below. A developer can place an element on a form to allow a user to input some data.

There are technologies available that allow for more complex elements by combining these elements together in a meaningful way. For example, a developer can use JSF [26] to create custom controls, or mixing HTML with JavaScript different input types are possible. At this stage, we have considered HTML 4.01 elements only though the approach is extensible and part of our future work.

[2] HTML 5.0 is not finalised but our approach could include new elements from the specification.

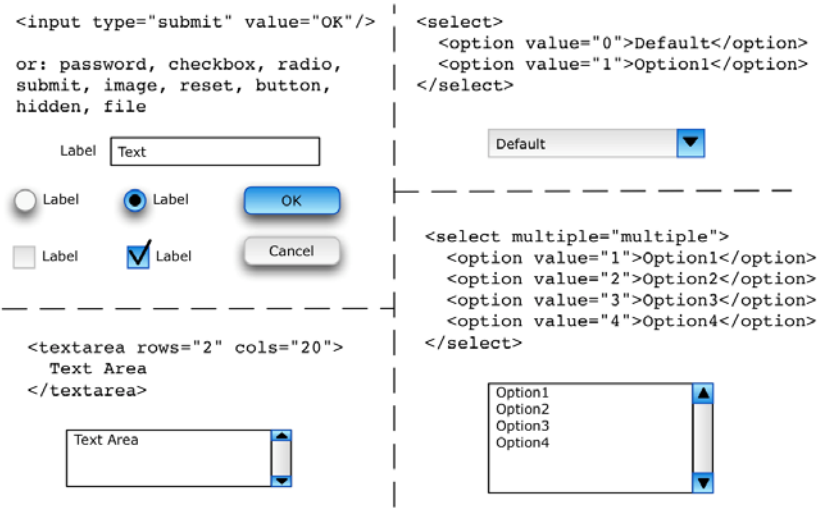

Fig. 2. HTML 4.01 form elements

4.2 Validation Categories

We conducted an analysis of several web sites to categorise different types of validation. These categories are used to determine how the validation will be included in the meta-models and eventually transformed into generated code for the target web application framework. The three categories we found are: *single element*, *multiple element*, and *entity association*.

Single element refers to validation that occurs on one HTML element only. For example, a text field must not be empty, or the integer entered must be less than ten. Depending on the type of HTML element, different web application frameworks provide built-in validation. All frameworks allow setting minima and maxima for numerical types and string lengths. They also all have some way to use regular expressions for validation; for example, email addresses, credit cards, dates, etc.

Multiple element validation occurs when the value entered by a user on one element has an effect on what is expected in another element. For example, if a checkbox is ticked then a text field must not be empty, or a date entered in an element must be before another date in a different element.

Entity association refers to the class relationships that exist in the domain. The domain is included in the majority of web modelling languages, e.g. in UWE it is known as a content model. UML class diagrams and the associations between the classes have a multiplicity, e.g. a one-to-many relationship. In this case, unlike a zero-to-many, it may be required that the one multiplicity is required.

All of the three validation categories are included in our example. Fig. 3 is a screen capture of the validation that occurs when a user attempts to save a new product without filling in any of the form. The error messages are displayed to the right of the element. The form has been generated and is part of a web application.

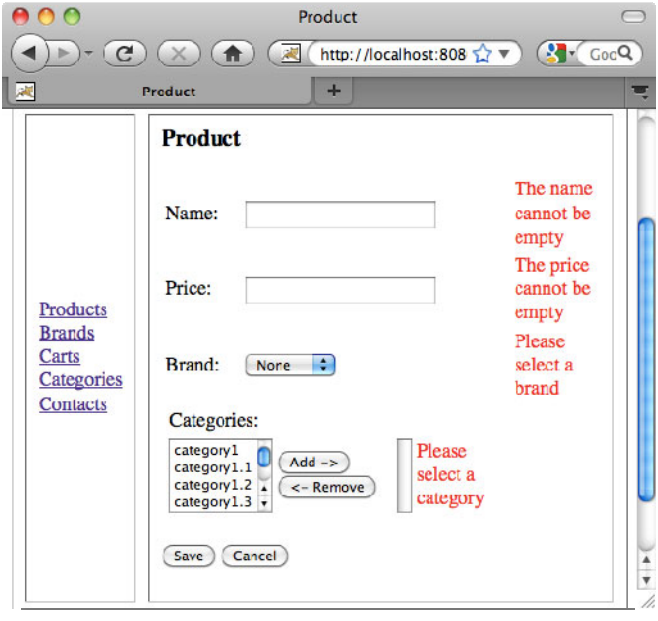

Fig. 3. Example of Product form validation

4.3 Modelling and Transformations

Our model-driven solution to web form validation is part of a solution that generates web applications using Spring MVC and Hibernate [27]. For the context of this section we will give a brief summary of our models and transformations. Since the majority of web application frameworks are based on MVC we create three models, one model for each of the MVC-triad. We name our models the same: *Model*, *View*, and *Controller*. These models form the basis of our graphical DSL for web applications.

We build our target application and then, through a process of abstraction, map the application to the models. Stahl and Völter [3] describe AC-MDSD alongside a two-track iterative development process and we have had success in generating web applications with this approach. A full description of our models and transformations is beyond the scope of this paper, although recognition of the included models is needed to understand this section.

Single Element. UML stereotypes are used to model single element validation. By analysing the target web application framework, in this case Spring using JSR:303, we identify that the annotations can be generated using UML stereotypes. For example, as shown in Fig. 4, the *Product's* name is stereotyped `<<NotEmpty>>` and the price is stereotyped `<<Min>>`. The `<<Min>>` stereotype has a property that is used in the transformation and, in this case, the developer has set the value to zero.

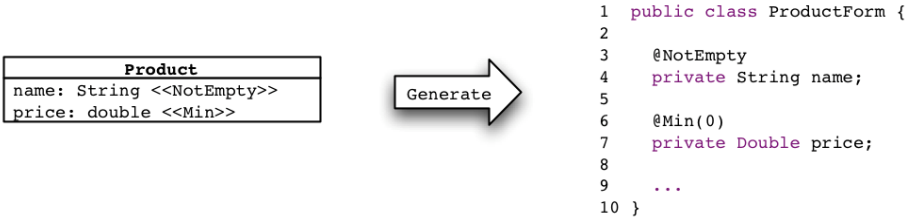

Fig. 4. Product with stereotypes

We use the Eclipse Modelling Project [28] and subprojects for our modelling and transformation track. The models are UML2 [29] with Profiles using JET [30] for the model-to-code transformations. JET uses templates and we implement a multi-stage generator that uses intermediate XML beans.

Multiple element. We use OCL constraints (invariants) to represent multiple element validation, as UML stereotypes do not provide sufficient flexibility. For example, a *Contact* must input an email address, a phone number, or both. The OCL invariant is:

```
email.size() > 0 or phone.size() > 0
```

We place these invariants on the UML class in the domain model. If an invariant is found during the transformations, we use the Eclipse subproject OCL [31] to assist generate the form validation code. OCL has an abstract syntax tree and we have implemented a *visitor* (pattern) that produces Java expressions. The following program code shows how the OCL constraint is created on line 1; and on line 3 the expression is visited by our OCL to Spring Visitor which outputs the required Java code.

```
1 OCLExpression query = helper.createQuery(expression);
2 OCL2SpringVisitor visitor = new OCL2SpringVisitor();
3 String code = query.accept(visitor);
```

The Java code is stored as a string and further along the transformation process it is passed to a JET template that is responsible for producing the Spring validator class shown below. The boolean expression for the if statement on lines 7 and 8 is the code produced by our visitor. It is the negation of the OCL constraint, which is the condition that should reject the value.

```
1  public class ContactValidator implements Validator {
2    public boolean supports(Class clazz) {
3      return ContactForm.class.equals(clazz);
4    }
5    public void validate(Object obj, Errors e) {
6      ContactForm form = (ContactForm) obj;
7      if((form.getEmail().length() <= 0) &&
8        (form.getPhone().length() <= 0)) {
9        e.rejectValue("contact.multiple");
10     }
11   }
12 }
```

The result of an OCL expression is true or false, making it well suited for form validation. By visiting an OCL expression we are able to create the equivalent Java expression needed to be placed in the web application custom validator. Eclipse OCL does model checking prior to our transformations ensuring that the OCL is syntactically correct and references valid attributes in a UML class diagram.

Entity Association. Enforcing multiplicities on class associations is dependent on the requirements of the web application. Some projects do not require this to be validated while others do. For our example, when an entity is either created or updated the association multiplicities are enforced. This entails checking the associations and if an association had a value of '1', we then add in validation. In Fig. 3, this can be seen as a *Product* has a *..1 association to a *Brand*. So, the user must select a brand before saving.

No additional UML stereotypes or OCL needs to be included. The UML class associations can have multiplicities and during the transformation we check for these. When one is found we add validation to the appropriate validator as shown in the following code. Line 7 checks that the user has selected a *Brand* other than the default value of '0'. Fig. 3 shows the validation working in a browser.

```
1  public class ProductValidator implements Validator {
2    public boolean supports(Class clazz) {
3      return ProductForm.class.equals(clazz);
4    }
5    public void validate(Object obj, Errors e) {
6      ProductForm form = (ProductForm) obj;
7      if(form.getSelectedBrand()[0].equals("0")) {
9        e.rejectValue("brand.atLeastOne");
10     }
11   }
12 }
```

4.4 UML Profile

Fig. 5 shows part of the UML Profile used for *single* and *multiple element* validation. All stereotypes inherit from *Bundle*, which has two properties: `errorMessage` and `errorCode`. In our generated web application, these are transformed into a resource bundle that is used by Spring MVC to display error messages to the user.

The *Single* stereotype is abstract and subclassed for each annotation of JSR:303. For brevity, Fig. 5 only shows some of the JSR:303 annotations. A *Single* stereotype can be applied to a `Property`, which is called an extension and is depicted by an arrow with the head filled in [1, p.659]. Similarly, the `Multiple` stereotype can be a applied to a `Class`. The extension restricts the UML elements that the stereotype can be applied to.

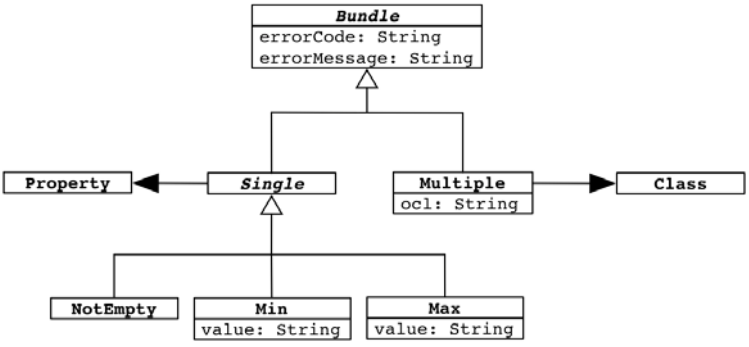

Fig. 5. Validation Profile

We have illustrated by example in this section how to apply our UML profile to a domain model. It is possible to have form elements that are not directly related to an entity from the domain model. These form elements may need to be validated. In WebDSL, this category of validation is called *input assertions.* We do not have a separate category, as we are able to apply the UML profile to our *View* model and reuse all of the same validation. It is beyond the scope of this paper to explain our *View* model in depth, but the UML profile enables us to apply validation to other models than the domain.

5 Discussion

Stahl and Völter [3] recommend that target artefacts are hand-written at least once in AC-MDSD. So, we build a *reference implementation* and then infer our model and transformations. If we wanted to include client-side validation we would start with making changes to the *reference implementation*. Once we were satisfied with the client-side validation we would then update our transformations and include any extra information necessary in the model. Similarly, for new HTML 5 elements we would include them following this same process.

So, if we were to follow the process again, but for a different web application framework, would we be able to reuse this approach with UML and OCL? Each of the four frameworks we analysed has some standard validators as shown in Table 1. Additionally, each framework provides custom validation whereby the developer can validate any of the form input. Therefore, it would be possible to use our approach again as *single element* validation can use UML stereotypes with a simple one-to-one mapping with the standard validators in Table 1. *Multiple element* validation and *entity association* would again need to be implemented via a custom validator.

The AC-MDSD goals of development efficiency, software quality, and reusability differ from the MDA goals of interoperability and software portability. MDA aims to achieve these goals via transformations from the platform independent model (PIM) to the platform specific model (PSM) and subsequently to the code. In our example, by mapping the web application framework standard validators to UML stereotypes

we are making our models specific to that web application framework, effectively creating PSMs. If we were interested in creating a PIM we would need a suitable approach that would cover the standard validators as seen in Table 1. Using OCL for *single element* validation could be a viable solution and this is part of our future work.

Table 1. Standard validators for web application frameworks. In addition, each framework has the ability to provide custom validation.

Framework	Validation
Spring	AssertFalse, AssertTrue, DecimalMax, DecimalMin, Digits, Email, Future, Length, Max, Min, NotNull, NotEmpty, Null, Past, Pattern, Range, Size, Valid
ASP.NET MVC	Range, RegularExpression, Required, StringLength
Ruby on Rails	validates_acceptance_of, validates_associated, validates_confirmation_of, validates_each, validates_exclusion_of, validates_format_of, validates_inclusion_of, validates_length_of, validates_numericality_of, validates_presense_of, validates_size_of, validates_uniqueness_of
Grails	blank, creditCard, email, inList, matches, max, maxSize, min, minSize, notEqual, nullable, range, scale, size, unique, url

6 Conclusion

Form validation is an important part of a web application and must be considered in model-driven web development. In this paper, we present an example of applying AC-MDSD using a two-track development methodology. Our models are UML and OCL compliant and the scope of our example is HTML 4.01 form elements with server-side validation. Client-side validation and complex form elements are future work.

We categorised validation into *single element*, *multiple element*, and *entity association*. We used UML stereotypes for single element and OCL for multiple element validation. Entity association is expressed as part of the domain model, which is a UML class diagram. In our transformations we checked the multiplicity of the associations and applied validation accordingly. We were able to successfully generate a Spring web application from our models and display them in a browser for manual testing.

Our analysis of web application frameworks included four frameworks with different programming languages. We observed some similarities between the different frameworks and this could be exploited to create a validation meta-model to be applied to more than one implementation. This approach does have an inherent risk that attempting to apply a general model for all implementations may miss some features of a framework. This issue will be addressed as part of our future work.

Acknowledgements. We would like to thank the Australian Postgraduate Award, The University of Queensland.

References

1. Unified Modeling Language (April 10, 2011), http://www.omg.org/spec/UML/2.2
2. Object Constraint Language (April 10, 2011), http://www.omg.org/spec/OCL/2.2
3. Stahl, T., Völter, M.: Model-Driven Software Development: Technology, Engineering, Management. John Wiley, Chichester (2006)
4. Model Driven Architecture, http://www.omg.org/mda
5. Bockle, G., Clements, P., McGregor, J.D., Muthig, D., Schmid, K.: Calculating ROI for Software Product Lines. IEEE Software 21(3), 23–31 (2004)
6. Valderas, P., Fons, J., Pelechano, V.: Transforming Web Requirements into Navigational Models: AN MDA Based Approach. In: Delcambre, L., Kop, C., Mayr, H.C., Mylopoulos, J., Pastor, Ó. (eds.) ER 2005. LNCS, vol. 3716, pp. 320–336. Springer, Heidelberg (2005)
7. Rossi, G., Schwabe, D.: Modeling and Implementing Web Application with OOHDM. In: Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.) Web Engineering: Modelling and Implementing Web Applications, pp. 109–155. Springer, Heidelberg (2008)
8. Hennicker, R., Koch, N.: A UML-Based Methodology for Hypermedia Design. In: Evans, A., Kent, S., Selic, B. (eds.) UML 2000. LNCS, vol. 1939, pp. 410–424. Springer, Heidelberg (2000)
9. Bolchini, D., Garzotto, F.: Designing Multichannel Web Applications as "Dialogue Systems": the IDM Model. In: Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.) Web Engineering: Modelling and Implementing Web Applications, pp. 193–219. Springer, Heidelberg (2008)
10. Ceri, S., Fraternali, P., Bongio, A.: Web Modeling Language (WebML): a modeling language for designing Web sites. Computer Networks 33(1-6), 137–157 (2000)
11. Houben, G.J., Sluijs, K., Barna, P., Broekstra, J., Casteleyn, S., Fiala, Z., Frasincar, F.: HERA. In: Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.) Web Engineering: Modelling and Implementing Web Applications, pp. 263–301. Springer, Heidelberg (2008)
12. Troyer, O.D., Casteleyn, S., Plessers, P.: WSDM: Web Semantics Design Method. In: Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.) Web Engineering: Modelling and Implementing Web Applications, pp. 303–351. Springer, Heidelberg (2008)
13. Kroiss, C., Koch, N., Knapp, A.: UWE4JSF: A Model-Driven Generation Approach for Web Applications. In: Gaedke, M., Grossniklaus, M., Díaz, O. (eds.) ICWE 2009. LNCS, vol. 5648, pp. 493–496. Springer, Heidelberg (2009)
14. Schmid, H., Donnerhak, O.: OOHDMDA – An MDA Approach for OOHDM. In: Lowe, D., Gaedke, M. (eds.) ICWE 2005. LNCS, vol. 3579, pp. 569–574. Springer, Heidelberg (2005)
15. Rutten, B., Barna, P., Frasincar, F., Houben, G.J., Vdovjak, R.: HPG: a tool for presentation generation in WIS. In: Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters, pp. 242–243. ACM (2004)
16. Acerbis, R., Bongio, A., Brambilla, M., Butti, S.: WebRatio 5: An Eclipse-Based CASE Tool for Engineering Web Applications. In: Baresi, L., Fraternali, P., Houben, G.-J. (eds.) ICWE 2007. LNCS, vol. 4607, pp. 501–505. Springer, Heidelberg (2007)

17. Apache Struts, http://www.struts.apache.org
18. Visser, E.: WebDSL: A Case Study in Domain-Specific Language Engineering. In: Lämmel, R., Visser, J., Saraiva, J. (eds.) Generative and Transformational Techniques in Software Engineering II. LNCS, vol. 5235, pp. 291–373. Springer, Heidelberg (2008)
19. Groenewegen, D.M., Visser, E.: Integration of Data Validation and User Interface Concerns in a DSL for Web Applications. In: van den Brand, M., Gašević, D., Gray, J. (eds.) SLE 2009. LNCS, vol. 5969, pp. 164–173. Springer, Heidelberg (2010)
20. Spring Framework (March 1, 2011), http://www.springsource.org
21. Ruby on Rails (March 1, 2011), http://www.rubyonrails.org
22. Grails (March 1, 2011), http://www.grails.org
23. ASP.NET MVC (March 1, 2011), http://www.asp.net.mvc
24. JSR: 303 (March 1, 2011), http://jcp.org/en/jsr/detail?id=303
25. HTML 4.0.1 Specification, http://www.w3.org/TR/html401/
26. Java Server Faces (March 1, 2011), http://java.sun.com/javaee/javaserverfaces/
27. Hibernate (March 1, 2011), http://www.hibernate.org
28. Eclipse Modeling Project (March 15, 2011), http://www.eclipse.org/modeling/
29. UML2 (March 15, 2011), http://www.eclipse.org/modeling/mdt/?project=uml2
30. JET (March 15, 2011), http://www.eclipse.org/modeling/mdt/?project=jet
31. OCL (March 15, 2011), http://www.eclipse.org/modeling/mdt/?project=ocl