A Model-Driven Approach for the Design of Web Information Systems based on Frameworks

Beatriz Franco Martins  
Ontology & Conceptual Modeling Research Group (Nemo) - Department of Informatics, Federal University of Espírito Santo (Ufes)  
Vitória, ES, Brazil  
bfmartins@inf.ufes.br

Vítor E. Silva Souza  
Ontology & Conceptual Modeling Research Group (Nemo) - Department of Informatics, Federal University of Espírito Santo (Ufes)  
Vitória, ES, Brazil  
vitor.souza@ufes.br

ABSTRACT

In the field of Web Engineering, many methods have been proposed. FrameWeb is a method that targets the development of systems that use certain kinds of frameworks in their architecture, proposing the use of models that incorporate concepts from these frameworks during design. However, in its original proposal, FrameWeb's models do not fit well different framework instances, its language is not formally defined and no tool support is offered to aid software architects in creating the models. In this paper, we propose to address these issues using model-driven techniques.

Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques—Object-oriented design methods

Keywords

Web Engineering; Frameworks; Model-Driven; FrameWeb; Meta-Model.

1. INTRODUCTION

Since Web Engineering (WebE) was born, many methods that support the construction of applications for the Web platform have been proposed. FrameWeb [14] is one such method. It targets the development of Web Information Systems (WISs) that use certain kinds of frameworks in their infrastructure (e.g., front controller, object/relational mapping, dependency injection). Given the popularity and their infrastructure (e.g., front controller, object/relational mapping systems that use certain kinds of frameworks in their original proposal, FrameWeb's models do not fit well different framework instances, its language is not formally defined and no tool support is offered to aid software architects in creating the models. In this paper, we propose to address these issues using model-driven techniques.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

WebMedia '15 October 27 - 30, 2015, Manaus, Brazil  
© 2015 Copyright held by the owner/author(s). Publication rights licensed to ACM.  
ISBN 978-1-4503-3959-9/15/10...$15.00  
DOI http://dx.doi.org/10.1145/2820426.2820439

2. BASELINE

In this section we summarize the original FrameWeb proposal and the basic concepts of Model-Driven Development.

2.1 FrameWeb

Software reuse has been practiced since programming began, using, e.g., libraries, domain engineering, design patterns, componentry, etc. [9]. A popular method of reuse is the use of software frameworks or platform architectures (e.g., Java™ Enterprise Edition [3]), which are middleware on/with which applications can be developed [1]. The use of framework helps to avoid the continual rediscovery and reinvention of basic architectural patterns and components, reducing cost and improving the quality of software by using proven architectures and designs [14].

Motivated by this, the Framework-based Design Method

In this paper, the term framework is used both in its traditional sense—a reusable set of libraries or classes for a software system—and in the sense of platform architectures mentioned above.
for Web Engineering (FrameWeb) [15] was proposed with the goal of handling the complexity behind implementing Web-based Information Systems (WISs) by incorporating concepts from well established frameworks into higher-level development artifacts, i.e., architectural design models. Table 1 shows the types of frameworks supported by the method.

FrameWeb proposes: (i) a basic architecture that divides the system in layers (Presentation, Business Logic and Data Access) for better integration with the type of frameworks shown in Table 1 and (ii) a UML profile for the construction of four different design models (all based on the UML Class Diagram) that bring the concepts used by these frameworks to the architectural design stage of the software process:

- **Persistence Model**: represents the Data Access Objects (from the DAO pattern [1]) responsible for the persistence of domain objects (Data Access layer);
- **Domain Model**: represents domain classes and their object/relational meta-data using concepts from the ORM framework (Business Logic layer);
- **Application Model**: represents the classes that are responsible for implementing the system’s functionalities (Business Logic layer) and their relationship with classes from other layers (i.e., the dependencies), using concepts from the DI framework;
- **Navigation Model**: represents the components that form the presentation layer, such as Web pages, HTML forms, etc. (Presentation layer), using concepts from the Front Controller (MVC) framework.

Due to space constraints, we illustrate only one of the models described above (for more details, refer to [15]). Figure 1 shows a Navigation Model taken from the architectural design of SCAP, an application that helps universities manage leave of absence requests from professors, built using Struts 2 as the MVC framework.

The model defines the architecture of the Presentation layer for one of SCAP’s functionalities—request leave of absence—following the constructs of Struts 2. Stereotypes (or lack thereof) define the type of each component. This particular scenario starts with the Web page `index.jsp` requesting the input of the submit method of the action (controller) class, which leads to the `form.jsp` page. This is shown by the dependency associations between `index.jsp` and `requestLeaveAction`, and between the latter and `form.jsp`.

The `form.jsp` page contains a form with six fields, represented as the attributes of `requestLeaveForm`, whose types are given according to the Struts 2 tag used in that particular field (e.g., `select` creates a drop-down menu; `textfield` is a one-line text field). According to a convention, the names of the attributes indicate that the contents of the form fields should be bound to specific attributes of the action class. For instance, `request.eventName` refers to the `eventName` attribute of the `request` object at the controller.

Once again following the dependencies between classes, when the form is submitted, the values of its fields are sent and the `executeSubmit()` method is called. If it returns `input`, we go back to the form (which happens when there are errors in the submitted data); if it returns `success`, it displays the `done.jsp` page instead. That last page refers to the `request` in its `text` component, to show that the leave of absence request has been successfully registered.

It is important to note that, although the Navigation Model focuses on behavior, a structural diagram (i.e., the Class Diagram) was chosen as basis for it instead of a behavioral one (e.g., Sequence or Activity diagrams). This choice was result of experiment with developers and the Class Diagram was chosen for it represents more adequately the internal structure of controllers, Web pages and forms.

### Table 1: Types of frameworks supported by FrameWeb.

<table>
<thead>
<tr>
<th>Type of Framework</th>
<th>Supported in [15]</th>
<th>Java EE Standard</th>
<th>Other Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Front Controller (MVC)</td>
<td>Struts 2</td>
<td>JSF</td>
<td>GWT, Spring MVC, VRaptor</td>
</tr>
<tr>
<td>Object/Relational mapping (ORM)</td>
<td>Hibernate</td>
<td>JPA</td>
<td>Cayenne, OpenJPA, pBeans</td>
</tr>
<tr>
<td>Dependency Injection (DI)</td>
<td>Spring Framework</td>
<td>CDI</td>
<td>Guice, PicoContainer, Plexus</td>
</tr>
</tbody>
</table>

### Figure 1: Example of FrameWeb Navigation Model.

2.2 Model-Driven Development

Model-Driven Development (MDD) [11] is a development approach that focuses in models: instead of building software using programming languages separately from the software design artifacts, MDD allows us to develop the functionalities of the system based on a set of models that, after going through transformations between different levels of abstraction, result in the final software system.

MDD gained a lot of attention from both industry and research communities because of its potential to provide a way for increasing productivity during application development and maintenance. In an MDD process, models are specified with a clear abstract syntax, with well-defined rules for their interpretation. This makes them more easily processable by...
machines, which in turn enables tools for model creation, transformation, validation, etc.

There are currently several model-driven tools and frameworks to aid software developers in following a model-driven approach. A very popular free and open-source tool is the Eclipse IDE and its components EMF, OBEo Designer, Acceleo, XText, among others. More details on the use of such tools will be given in Section 3.3.

3. PROPOSAL

In this section we present our proposal for applying MDD tools and techniques to FrameWeb. Because of paper size and organization concerns, we keep the focus only on FrameWeb’s Navigation Model. However, the entire work can be found at the FrameWeb project website.

The contributions of this work are: (a) a meta-model for FrameWeb, based on UML 2.0, serving as the formal abstract syntax for the FrameWeb language, evolving the original UML profile[15]; (b) a flexible approach which allows different frameworks instances to be used with FrameWeb, enabling designers to choose the most appropriate combination of frameworks in their context; (c) a FrameWeb design tool prototype based on the meta-model which can help software architects apply the method to the development of Web Information Systems. We present these contributions next.

3.1 Meta-model

We propose to evolve FrameWeb from a lightweight to a full UML extension, defining a Domain-Specific Language (DSL) for the method. We start from the UML meta-model and add domain-specific types to build the FrameWeb meta-model. We chose this approach to be able to treat the various specific aspects related to frameworks.

We define the FrameWeb Meta-model at a higher meta-level (M2) and depending on the UML Meta-model, as shown in Figure 2. However, since our meta-model refers only to a small part of UML, we create an intermediary package called Partial UML Meta-model which shows (graphically) only the UML meta-classes that are actually used and how they relate to FrameWeb meta-classes, facilitating visualization by developers. The FrameWeb Meta-model then specifies the syntax of the FrameWeb language, which drives the creation of its models. We can break the syntax into two parts: framework-independent and framework-dependent syntax.

The framework-independent syntax drives model creation under a general point of view, i.e., independently of the specific framework instances being used. This means that all models created at level M1 using the FrameWeb language need to follow the same platform-independent rules. For instance, in a Navigation Model the only types of classes allowed are Navigation Classes.

For each FrameWeb model (Section 2.1), there is a meta-model part which defines its syntax, also shown in Figure 2: the Domain Model is an instance of the Domain Meta-model part, the Persistence Model is an instance of the Persistence Meta-model part, and so on. All four parts depend on the Framework Meta-model part, which specifies the framework-dependent syntax (discussed in the next section).

In Figure 3 we depict a fragment of the Navigation Meta-model, which defines the syntax of Navigation Models. Its meta-classes represent components of the Presentation layer.

For instance, the meta-classes Page, Form and FrontControllerClass represent, respectively, a dynamic Web page, a form in such a page and the controller class.

In Figure 4, we show an instance of this meta-model, representing the same functionality of SCAP previously shown in Figure 1 but using JSF as the Front Controller instead of Struts 2. The stereotypes from the original profile of FrameWeb are still used to identify the meta-class of each class in the model, e.g., Page ( stereotype <<page>>), Form ( stereotype <<form>>) and FrontControllerClass (no stereotype).

The model of Figure 4 shows two pages (form.xhtml and success.xhtml), a form (requestLeaveForm) and a controller class (RequestLeaveController). The meta-model (M2) specifies how these components (in M1) can relate to one another, allowing only some specific dependencies and compositions relations between them.

For instance, form components can submit their data to controller classes, which is represented by dependency relations between these components, defined by the FrontCon-
controllerDependency meta-class (Figure 3). In the example (Figure 4), the form requestLeaveForm submits its data to the RequestLeaveController class and calls the submit() method (FrontControllerMethod meta-class).

Analogously, controllers can direct results to Web pages, also through dependency relations defined by the ResultDependency meta-class. RequestLeaveController sends the result back to form.xhtml if the result is null or to success.xhtml if it is "success.xhtml" (Result meta-class).

Moreover, Web pages can be composed of forms, represented by a composition association and defined by the NavigationComposition meta-class (between Page and Form meta-classes). In our example, form.xhtml is composed of requestLeaveForm. Finally, forms are themselves composed of form fields, which appear as class attributes in FrameWeb’s Navigation Models. In the meta-model these are represented by the meta-composition between the meta-classes Form and FormComponent. In SCAP, requestLeaveForm is composed of two drop-down menus (selectOneMenu) and four text fields (inputText).

In addition to the FrameWeb abstract syntax defined in its meta-model, there is a set of rules written in Object Constraint Language (OCL). Because of limitations in UML Class Diagram expressiveness many rules are necessary to define the correct syntax for the FrameWeb language.

We present here, however, only a simple rule example that illustrates the role of OCL in the meta-model (other examples can be accessed via the project website). In the context of a ResultDependency, the dependency client must be a FrontControllerClass and the supplier must be NavigationClass or a FormComponent. No other UML classes are allowed to participate of this kind of dependency. This is defined in the following OCL invariant:

\[
\text{context ResultDependency}
\text{inv:}
\text{if (self.oclAsType(Dependency).client.oclIsTypeOf(FrontControllerClass)) and}
\text{((self.oclAsType(Dependency).supplier.oclIsTypeOf(NavigationClass))) or}
\text{((self.oclAsType(Dependency).supplier.oclIsTypeOf(FormComponent)))}
\]

It is important to note that the Navigation Model for the Struts 2 version of the example shown in Figure 1, although referring to the original proposal of FrameWeb, also seems
to follow the syntax defined by the meta-models proposed in this section. This, however, is merely due to the discipline of the architect that produced the model, given that in its original proposal the syntax of the language was only informally defined and there were no rules to constrain or support the designer.

### 3.2 Multi-Framework Approach

The framework-independent syntax presented in the previous section applies to all models built using the FrameWeb language, independently of the specific set of frameworks included in the system’s architecture. However, different framework instances have different characteristics that influence the models. For instance, Struts 2 form fields are bound to attributes in a single controller class, whereas in JSF one can bind fields from a single form to attributes in different controller classes.

Therefore, we need an approach that allows us to accommodate different framework instances. Each MVC framework, for example, has its own set of tags, in addition to the standard HTML and XHTML tags, so the FrameWeb language must be able to express and accept different framework tag syntax. We thus define a Framework Meta-model part, depicted in Figure 5. As seen before, the four generic FrameWeb meta-model parts depend on the Framework Meta-model, thus including framework-dependent rules in the different models of FrameWeb.

The TagLib meta-class represents the tag libraries provided by the chosen framework (e.g., the HTML tag library of JSF). Each taglib has a set of tags (e.g., the selectOneMenu and inputText tags used in Figure 4), which are represented by the Tag meta-class. Once a page imports a specific taglib, any tags from that library can be used by the page and all of its parts (e.g., its forms). In the Navigation Model, tag classes (i.e., instances of the Tag meta-class) from the chosen framework’s taglibs are used as types of page and form attributes.

As frameworks have different applications according to their type, the FrameworkCategoryList meta-class represents the kinds of frameworks covered by FrameWeb. This list is used to separate and drive the loaded frameworks behavior under the FrameWeb abstract syntax perspective.

Finally, the Rule meta-class allows the definition of a set of OCL rules to be loaded with each specific framework. In other words, each framework has its own constraints, and these need to be observed during the creation of the models. For instance, some frameworks prescribe naming conventions that must be followed in order to activate specific features, whereas other frameworks do not have this kind of constraint. Hence, it is necessary to load the OCL rules to guarantee the model is correct according to the chosen frameworks.

In order to accommodate a new instance of an MVC framework, we instantiate the meta-model of Figure 5 specifying the taglibs, tags and rules of that particular framework. For example, to be able to design SCAP using JSF, we define the JSF library depicted in Figure 6 in a file called MVC_JSF.frameweb. This file defines the taglibs: JSFcore (defined in the namespace http://java.sun.com/jsf/core), JSFhtml (http://java.sun.com/jsf/html) and JSFcomposite (http://java.sun.com/jsf/facetlets).

At design-time, the software architect loads the file corresponding to the frameworks she wishes to use. Figure 4 shows a Navigation Model for our running example using the prototype tool presented in the next section. We can see the MVC_JSF.frameweb file loaded into the editor at the bottom part of the screen, making the JSF taglibs available for the architect. At the center of the screen, we see the form component request.eventName from requestLeaveForm defined as a tag of kind inputText, a tag from the JSFhtml taglib.

Using this approach, other framework instances can be defined and loaded into any FrameWeb project—e.g., we can create a new SCAP Navigation Model using VRaptor, by loading its library and following the constraints of that framework. This approach can be used not only with MVC frameworks and Navigation Models, but with all FrameWeb models and any desired framework combination. In other words, FrameWeb is now extensible.

### 3.3 FrameWeb Tool

We propose a FrameWeb tool prototype capable of providing the necessary resources for the design of its models.
and, in addition, validate them. We used EMF, along with Eclipse plug-ins Sirius, OBEO Designer, Acceleo, UML 2.0 and OCL Tools for their UML 2.0/OCL support and code generation tools.

Firstly we implemented the FrameWeb meta-model using the Ecore Tools. Then, the FrameWeb Tree Editor is generated by EMF. This generation process has three main steps: (1) **Model source code generation**, which produces the necessary meta-classes according to the defined meta-model, using some validation features; (2) **Edit source code generation**, which performs command-base editing and has adapters to provide a structured view of the model code; and (3) **Editor source code generation**, which provides the editor user interface based on the EMF platform. The Model part is the core of the tool because it defines the FrameWeb language (abstract syntax) including all OCL rules.

EMF, however, is not able to automatically generate all source code, as the OCL rules require special handling. To deal with this, there are two different approaches: (a) manually add or update the validation classes to meet each OCL rule; or (b) use Eclipse tools for the automatic generation of the necessary validation classes, namely: the Java Emitter Templates (JET) tool, which allows us to create a set of general templates which are used as a frame under which EMF assembles the OCL rules; and the Java Merge (JMerge) tool used to produce the final code.

Following the latter (tools) approach, we write a set of validation class templates for the FrameWeb tool (*.javajet files). Then, the EMF Model generation automatically produces code for all OCL rules of the meta-model.

Finally, we obtained a FrameWeb Tree View Editor tool through which we can develop and, most importantly, validate FrameWeb models as shown before in Figure 7.

### 4. EVALUATION

To evaluate our work, we conducted an experiment with undergraduate students in which a WIS—the running example used in this paper, SCAP—was developed using different sets of frameworks. Based on the same requirements specification, the students produced FrameWeb models for their version of the system and implemented them using the selected frameworks.

Based on the reports produced by the students, this experiment allowed us to: (a) exercise the original FrameWeb method using different framework instances than the ones used in [15]; (b) evaluate what are the common aspects of the used frameworks (i.e., framework-independent) and what are their differences (framework-dependent); (c) discover modeling constructs that FrameWeb was not yet able to support, thus eliciting requirements for the FrameWeb language proposed here; and (d) validate the models produced by the students using the Tree View Editor.

Figure 8 presents yet another version of the Navigation Model for the SCAP request leave of absence feature, to illustrate how the same use case can be designed with FrameWeb using different frameworks, resulting in distinct implementation but with a uniform syntax. We can see that different tags have been used in the form and that the framework imposes a slightly different flow than JSF (closer to Struts 2). Also, note that one of the pages shares part of the name with the controller class, which is due to a naming convention imposed by VRaptor.

---

4 These reports are also available at the FrameWeb website.
Figure 9: The SCAP VRaptor model created using the FrameWeb prototype tool.

By using the FrameWeb tool, the architect can verify if the designed model is valid, i.e., if it correctly follows the FrameWeb syntax, including the framework-dependent rules. In Figure 10, we purposely introduce an error in the model by not filling in the value for the mandatory property PageTagLib of header.xhtml. As a result, the tool displays an error message, shown in Figure 11.

5. RELATED WORK

The major motivation for the proposal of FrameWeb is the fact that although there are several Web Engineering methods defining languages and tools for the development of WISs, to our knowledge none of them focus on the important role of frameworks in the system architecture.

IFML [2], standardized by the Object Management Group (OMG) in 2013, is a visual, platform-independent language based on a traditional MVC approach. Using MDD techniques, it performs automatic code generation from diagrams, which are mainly focused on the user’s point of view and require prior knowledge of the proposed language. FrameWeb, on the other hand, requires less effort to learn its language, since it is based on UML.

Purificação and Silva [13] propose a DSL, called EngenDSL, which aims to build a declarative abstraction for building WIS, in order to avoid typical programming constructs (like conditional structures and loops) and commitment to a specific technology. It uses a traditional MVC approach, but does not consider the different framework instances, resulting in a project not necessarily reflecting the real implementation. FrameWeb considers the specific framework’s rules, representing more accurately what is in fact implemented.

OOH4RIA [9] is an extension to the OOH method with a model-driven approach, which performs M2M transformations from a Platform-Independent Model (PIM) to a Platform-Specific Model (PSM). The proposal is focused specifically on Google Web Toolkit (GWT) framework for the Java platform. In the same line, OOWS [4] propose PIM to PSM, M2M transformations, but the implementation uses OlivaNova Transformation Engine for the PHP platform. Instead, FrameWeb is meant to be flexible, allowing for any framework (of the supported kinds) to be included, even in platforms other than Java EE.

Jurista et al. [10] propose a framework to deal with Usability Features in an MDD method based on the concept of Mode of Use (MoU). Based on (and partly dependent of) the OO-Method [12], the framework tackles MDD deficiencies related to usability (according to the ISO standard 9241-11). As a design method, FrameWeb can be integrated to this proposal by adding the necessary meta-model elements which would allow us to define MoUs in our models. Thus, this proposal can be seen as complementary to our work.

Other works have used the MDD approach in Web Engineering, but targeting specific concerns, such as communication and collaboration [6], multimedia [7], or accessibility [8]. Our use of MDD targets the use of frameworks.

6. CONCLUSIONS

In this paper, we have evolved the FrameWeb method using model-driven tools and techniques, dealing with some of the limitations of the original approach. The new FrameWeb has a well-defined syntax based on an extensible

Figure 11: Example of validation error message.

Other works have used the MDD approach in Web Engineering, but targeting specific concerns, such as communication and collaboration [6], multimedia [7], or accessibility [8]. Our use of MDD targets the use of frameworks.

6. CONCLUSIONS

In this paper, we have evolved the FrameWeb method using model-driven tools and techniques, dealing with some of the limitations of the original approach. The new FrameWeb has a well-defined syntax based on an extensible
meta-model, which allows developers to include support for new instances of the kinds of frameworks supported by the method. Moreover, we provide a simple, but useful tool that allows architects to design FrameWeb models and validate them against general and framework-dependent rules.

This research is a work in progress and has many limitations that are subject to future work. Here, we highlight some of the more urgent ones.

The FrameWeb tool is still very simple, offering only the Tree View Editor and model validation. More advanced model-driven tools can be used to provide a graphical UML editor, the ability to import models built in other UML editors (to validate FrameWeb rules) and code generators, relieving developers of much of the coding effort. The editor presented here is currently being used as basis for the development of a more flexible and user-friendly graphical editor.

More experiments need to be conducted to assess the usefulness and effectiveness of FrameWeb. More instances of frameworks—including more kinds of frameworks (e.g., authentication & authorization, Aspect-Oriented Programming, etc.)—need to be tested to verify the completeness of the meta-model. Since each framework has its own characteristics, introduced via meta-model loaded libraries, each library must be checked and validated by themselves as well as their implementation/integration with the framework-independent meta-model.

More practitioners, preferably from outside the academic environment, should evaluate the method in more varied scenarios than a single system’s use cases (SCAP). Also, more attention should be given to activities of the software process other than design: how does the use of FrameWeb relates to Requirements Engineering, Testing, etc.? In terms of experimental Software Engineering, there is still much to be accomplished in our future work.

Finally, the choice of extending the UML meta-model, although it provides some benefits (familiar language to developers, tool support, etc.), is not set in stone. We intend to experiment with defining a DSL from scratch in order to compare pros and cons of each model-driven approach.

7. ACKNOWLEDGMENTS

This work has been directly supported by the 2015 call of the FAP institutional funding program from the Federal University of Espirito Santo. Nemo (http://nemo.inf.ufes.br) is currently supported by Brazilian research agencies CAPES and CNPq, process numbers 402991/2012-5, 485368/2013-7 and 461777/2014-2. We would like to thank our colleagues at Nemo for their feedback regarding this work and, in particular, prof. João Paulo A. Almeida and Cássio C. Reginato for their direct assistance in parts of it.

8. REFERENCES


