A Model-Driven Approach for Code Generation for Web-based Information Systems Built with Frameworks

Nilber Vittorazzi de Almeida
Ontology and Conceptual Modeling Research Group (NEMO)
Department of Computer Science, Federal University of Espírito Santo (UFES), Brazil
nilber@vittorazzi.com

Silas Louzada Campos
Ontology and Conceptual Modeling Research Group (NEMO)
Department of Computer Science, Federal University of Espírito Santo (UFES), Brazil
slcampos@inf.ufes.br

Vítor E. Silva Souza
Ontology and Conceptual Modeling Research Group (NEMO)
Department of Computer Science, Federal University of Espírito Santo (UFES), Brazil
vitorsouza@inf.ufes.br

ABSTRACT

In the field of Web Engineering, there are several methods proposed for the development of Web-based information systems (WISs). FrameWeb is a method that aims to develop WISs that use certain types of frameworks in their architecture, proposing models that incorporate concepts of these frameworks during system design. The method's modeling language is based on Model-Driven Development techniques, making it extensible to support different frameworks and development platforms. In this paper, we present a code generation tool for FrameWeb which harnesses the method's extensibility by being based on its language's meta-models. The tool works with an associated visual editor for FrameWeb models and showed promising results in initial evaluation efforts.

CCS CONCEPTS

• **Software and its engineering → Software design engineering**; *Object oriented frameworks*; • **Information systems** → Web applications;

KEYWORDS

Web Engineering; Frameworks; Model-Driven Development; Code Generation; FrameWeb

1 INTRODUCTION

Every year, there are new demands for the development of bigger and more complex software systems [11]. As consequence, there is an increasing need to create or improve tools and methods for software reuse, such as, e.g., frameworks [4], to reduce development time, keeping, however, the control and organization expected from the application of Software Engineering techniques.

In this context, the FrameWeb method [12] was proposed to aid a particular niche of developers, working on the construction of Web-based information systems (WISs) which include, in their architecture, certain types of frameworks which are very popular in Web development, namely: front controller, dependency injection and object/relational mapping frameworks. The models proposed by the method incorporate concepts used by these frameworks, bringing them to the architectural design phase of the software process, helping manage the complexity behind the implementation of the WIS and improving the communication between software architects and programmers.

By explicitly representing concepts from the frameworks in these models, programmers know exactly how to implement the various code artifacts to work with their respective frameworks. Many of these codification tasks, however, could be automated by code generation, relieving programmers from most of the coding effort and allowing them to concentrate more on business logic and less on infrastructure. There are many commercial code generation tools available, even free of charge, for Web applications based on frameworks. However, these tools are generally limited to one or a few specific frameworks and cannot be easily extended to support new frameworks or platforms. Existing research (cf. Section 5) makes use of the same techniques used by FrameWeb to promote extensibility but, unlike FrameWeb, are not focused on the role of frameworks in the system's architecture.

In this paper, we propose a code generation tool for FrameWeb, which generates code artifacts integrated with the set of frameworks selected by developers based on the WIS's architectural models. Just like FrameWeb, the code generator can be extended to support other frameworks and platforms, keeping the unified language proposed by the method, thanks to the meta-models that specify its modeling language. The code generator is an important step in the evolution of the approach, as it makes it possible to convert high-level Software Engineering artifacts (models) into code, contributing to an important phase of the software process.

The rest of the paper is divided as follows: Section 2 introduces research used as baseline in our work; Section 3 presents the FrameWeb code generation tool (and the associated visual editor for FrameWeb models), explaining how it works and how it can be extended; Section 4 reports on the tool's evaluation efforts; Section 5 compares our proposal with related work; finally, Section 6 presents the conclusions.

2 BASELINE

This section summarizes the FrameWeb method and the basic concepts of Model-Driven Development, on top of which we built our proposals in this paper.

WebMedia '17, October 17–20, 2017, Gramado, Brazil

ACM ISBN 978-1-4503-5096-9/17/10…$15.00
https://doi.org/10.1145/3126858.3126863

2.1 Model-Driven Development

Model-Driven Development (MDD) [8] is an approach that proposes that software development, maintenance and evolution be conducted using models, which can be validated and transformed from one stage to another, until the level of code.

In an MDD process, models are specified with a clear abstract syntax, with well-defined rules for its interpretation, given by meta-models. MDD models are, thus, more easily processed by machines; are independent of software (as code written in high-level programming languages is independent from hardware); can be transformed into code in several different programming languages; and can be partially or completely reused in different contexts.

MDD allows developers with different levels of individual experience to work together in a project; maximizes the work effectively done in the process; minimizes the workload needed to produce software, which can be validated by end users in shorter time-frames (i.e., they validate the models which generate the system); among other advantages. For these reasons, the FrameWeb method has been evolved from its original proposal [12] to use model-driven tools and techniques [6].

2.2 FrameWeb

FrameWeb [6, 12] is a method for the development of Web-based information systems (WISs) which use frameworks in their architecture. The approach proposes a basic architecture which divides the system into three main tiers (Presentation, Business Logic, and Data Access) for better integration with three types of frameworks: Front Controller (e.g., JavaServer Faces[1]), dependency Injection (e.g., Contexts and Dependency Injection for Java[2]) and Object/Relational Mapping frameworks (e.g., Java Persistence API[3]).

Moreover, the method proposes a UML-based modeling language for the construction of four different types of models (all of them based on UML's Class Diagram), which bring concepts used by the aforementioned frameworks during software implementation to the architectural design phase:

- **Persistence Model:** represents Data Access Objects (c.f. the DAO design pattern [1]), responsible for the persistence of domain objects (Data Access tier);
- **Entity Model:** represents domain classes and their metadata for object/relational mapping (Business Logic tier);
- **Application Model:** represents the classes responsible for the implementation of the system's functionality (Business Logic tier) and their relation with classes from other tiers (i.e., the specification of dependency injections);
- **Navigation Model:** represent the artifacts that compose the user interface, such as Web pages, forms, etc. and their integration with the Front Controller (Presentation tier).

Following an MDD approach, the abstract syntax of the method's Domain Specific Language (DSL) is specified by meta-models, which are based on parts of the UML meta-model, as shown in Figure 1. Each FrameWeb model has its syntax defined by a meta-model, e.g., the *Persistence Model* is an intance of the *Persistence Meta-model*, the *Entity Model* is an instance of the *Entity Meta-model*, etc.

[1]JSF, http://jcp.org/en/jsr/detail?id=344
[2]CDI, http://jcp.org/en/jsr/detail?id=346
[3]JPA, http://jcp.org/en/jsr/detail?id=338

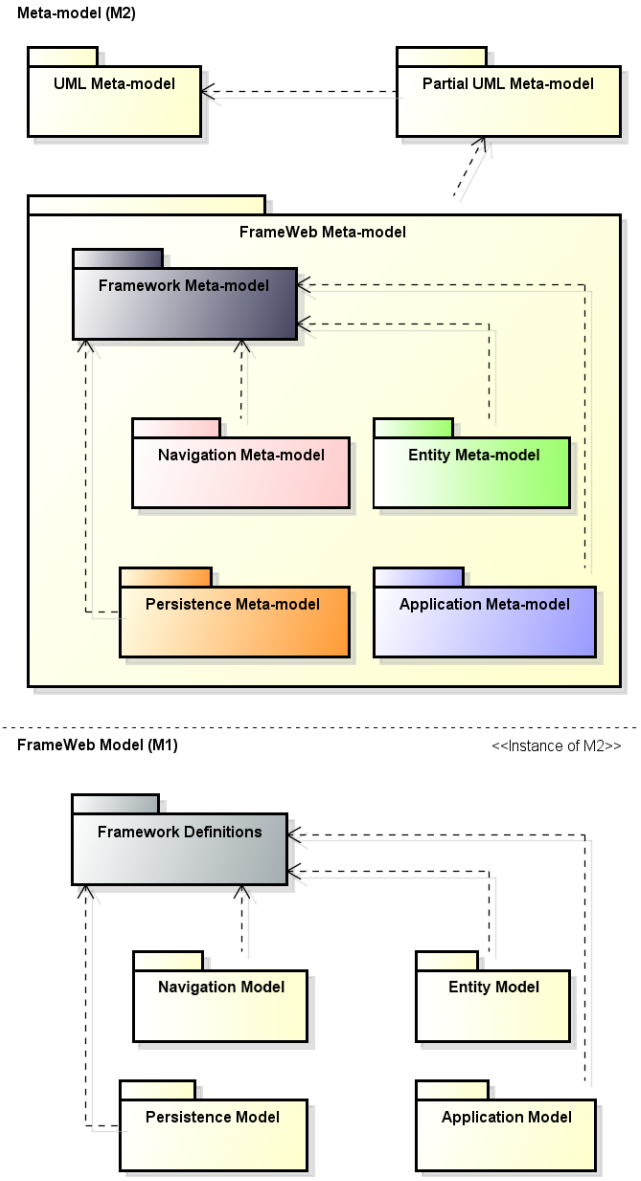

Figure 1: M1 and M2 abstraction levels for FrameWeb [6].

The meta-models corresponding to these four models are framework-independent, meaning that the models at level M1 follow a set of generic rules independent of the implementation platform chosen (e.g., regardless if the Front Controller framework used in the architecture is JSF or VRaptor[4]). The *Framework Meta-model*, then, allows developers to specify the *Framework Definitions*, which add to the models all the framework-dependent elements and rules. Hence, FrameWeb's modeling language is extensible, allowing developers to add support to new framework instances by creating new *Framework Definitions*.

[4]http://www.vraptor.org.

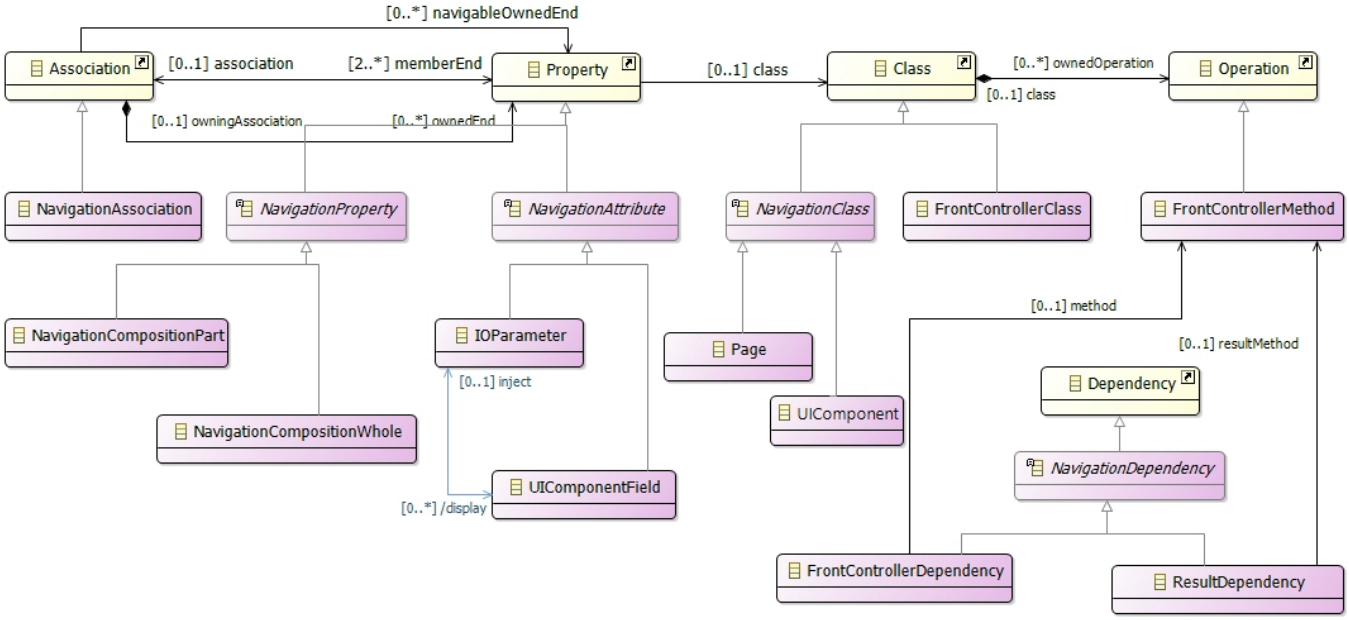

Figure 2: A fragment of FrameWeb's *Navigation Meta-model.*

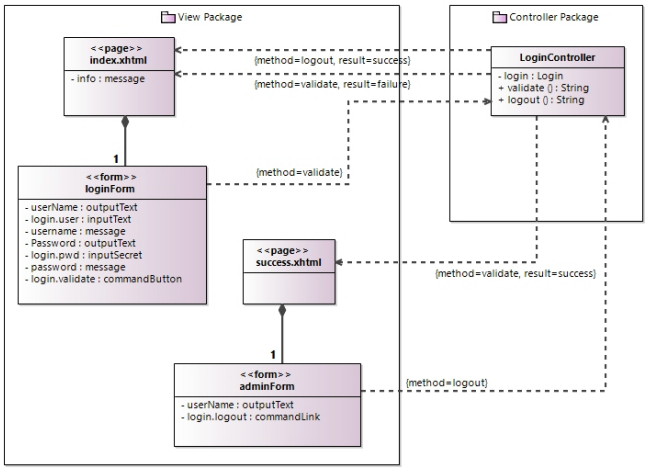

Figure 3: FrameWeb Navigation Model using JSF.

Figure 2 displays a fragment of the *Navigation Meta-model*, which defines the syntax of *Navigation Models*. Its meta-classes represent components of the Presentation tier. For instance, meta-classes `Page`, `UIComponent` and `FrontControllerClass` represent, respectively, a dynamic Web page, an HTML form and a controller class that mediates the communication between the page/form and the classes in the Business Logic tier.

An instance of this meta-model (i.e., a *Navigation Model*) is presented in Figure 3 and is used as a running example in this paper. The model represents a system that uses JSF as Front Controller framework,[5] i.e., the *Framework Definitions* for JSF have been imported here. Stereotypes are used to identify the meta-class of each class in the model, e.g., `Page` (`<<page>> stereotype`), `UIComponent` (`<<form>>` stereotype), and `FrontControllerClass` (no stereotype).

The model represents two Web pages (`index.xhtml` and `success.xhtml`), their forms (`loginForm` and `adminForm`, connected to their respective pages by a composition association) and one controller class (`LoginController`). The meta-model (M2) specifies how these components (in M1) can relate to one another, allowing some specific dependency and composition associations among them, with well-defined meanings.

This and the other models proposed by FrameWeb specify exactly how programmers should implement the interactions among these artifacts and their relation with the framework (in the case of Figure 3, the Front Controller framework). One could, however, relieve programmers from most of the coding effort by generating code based on these models. We present such a proposal next.

3 PROPOSAL

In this section, we present a visual editor and a code generator for FrameWeb, both of them based on the same Model-Driven Development techniques proposed by the method [6] and, thus, extensible to different frameworks and platforms. Interested readers can obtain their source code and meta-models at the project's website[6].

Subsection 3.1 introduces the graphical editor capable of drawing models following the languaged defined by FrameWeb. Subsection 3.2 presents the tool that generates code using such models as input. Finally, Subsection 3.3 describes the changes made in the meta-models proposed in [6] which allowed the development of these tools.

[5]This model was built based on an example extracted from the tutorial found at the website http://www.thejavageek.com/2013/12/18/login-application-jsf/.

[6]http://nemo.inf.ufes.br/projects/frameweb/.

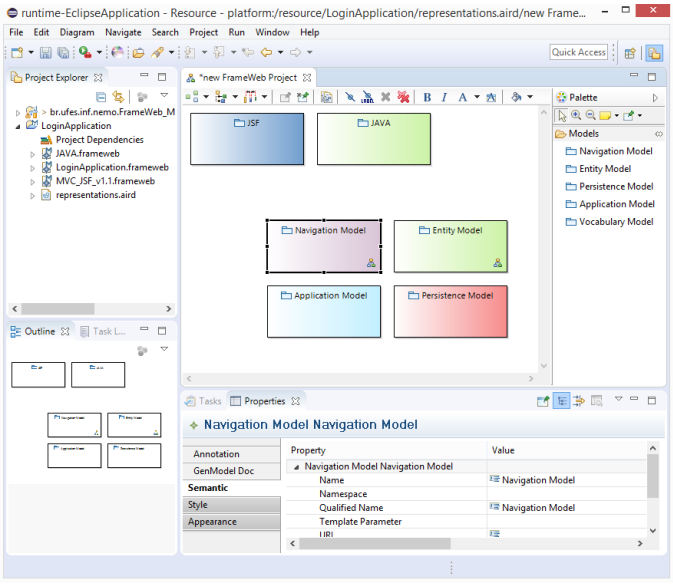

Figure 4: Project-level view for a WIS in *FrameWeb Editor*.

3.1 The FrameWeb Editor

In its original proposal [12], FrameWeb models were created using any UML editor, by using the UML Profile proposed by the method. The main limitation of this approach is the fact that a UML Profile is not rigorous, therefore UML modeling tools cannot prevent modelers from including elements that do not belong to the (FrameWeb) models and the fact that components of the model are not related to a specific syntax proposed by the method, hence they cannot be processed and interpreted by tools to provide useful features such as, e.g., code generation.

Using MDD techniques and tools, the FrameWeb meta-models presented in Section 2 served as basis for the development of a tool called *FrameWeb Editor*, which provides a graphical editor for the creation of valid models in this language. Such models are then used as input for the code generator proposed in this paper. Therefore, the *FrameWeb Editor* provides what is lacking in general-purpose UML tools in the context of FrameWeb, only allowing components to be created if they belong to the FrameWeb language and relating the constructs of the diagram directly to the language's syntax.

The *FrameWeb Editor* supports the creation of the four basic models proposed by the method: *Entity*, *Persistence*, *Application* and *Navigation* models, plus a project-level diagram that aggregates these four parts as well as *Framework Definitions* imported to the project. Figure 4 shows the main user interface of the tool, with the project-level view for the login application illustrated earlier in Figure 3 at the center of the editor. At the right-hand side, a panel shows the different components that can be added to the diagram at hand. At the bottom, there is a list of properties of the selected component in the diagram, allowing the user to view and change its attribute values. Finally, at the left-hand side, the *Project Explorer* displays an overview of the project, which, in the case of Figure 4, uses the Java platform and JSF as Front Controller framework.

Models created with the tool are saved in XML format with `.frameweb` extension and can be fed to the code generator proposed in this paper. Listing 1 displays a fragment of the XML code for our running example, shown earlier in Figure 3. In the code, we can identify different elements of the model shown in the figure, such as, e.g., the `packagedElement` tag with name `index.xhtml` and type `frameweb:Page`, corresponding to one of the Web pages in the model. The reader can relate other tags in the listing with elements of Figure 3 by looking at the `name` attribute of each tag and matching with components of the figure.

Listing 1: Fragment of `.frameweb` file created by the *FrameWeb Editor*, used as input for code generation.

```
<compose xsi:type="frameweb:NavigationModel" name="
    ↪ Navigation Model">
 <packagedElement xsi:type="frameweb:ViewPackage"
     ↪ name="View Package">
  <packagedElement xsi:type="frameweb:Page" name="
      ↪ index.xhtml">
   <ownedAttribute xsi:type="frameweb:
       ↪ UIComponentField" name="info">
    <type xsi:type="frameweb:Tag" href="MVC_JSF_v1.1.
        ↪ frameweb#//@configures.0/JSFhtml/message"/
        ↪ >
   </ownedAttribute>
   <ownedAttribute xsi:type="frameweb:
       ↪ NavigationCompositionWhole" type="//
       ↪ @compose.0/View%20Package/loginForm"
       ↪ association="//@compose.0/View%20Package/
       ↪ @packagedElement.5"/>
  </packagedElement>
  <packagedElement xsi:type="frameweb:Page" name="
      ↪ success.xhtml">
   <ownedAttribute xsi:type="frameweb:
       ↪ NavigationCompositionWhole" type="//
       ↪ @compose.0/View%20Package/adminForm"
       ↪ association="//@compose.0/View%20Package/
       ↪ @packagedElement.3"/>
  </packagedElement>
  <packagedElement xsi:type="frameweb:UIComponent"
      ↪ name="adminForm">
   <ownedAttribute xsi:type="frameweb:
       ↪ UIComponentField" name="userName"
       ↪ visibility="private">
    <type xsi:type="frameweb:Tag" href="MVC_JSF_v1.1.
        ↪ frameweb#//@configures.0/JSFhtml/
        ↪ outputText"/>
   </ownedAttribute>
   <ownedAttribute xsi:type="frameweb:
       ↪ UIComponentField" name="login.logout"
       ↪ visibility="private">
    <type xsi:type="frameweb:Tag" href="MVC_JSF_v1.1.
        ↪ frameweb#//@configures.0/JSFhtml/
        ↪ commandLink"/>
   </ownedAttribute>
  </packagedElement>
```

Listing 2: Fragment of the `App.Config` configuration file.

```
<appSettings>
  <add key="dir_template" value="template\"/>
  <add key="project" value="java_jsf"/>
  <add key="lang" value="java"/>
  <add key="dir_output_web" value="WebContent"/>
  <add key="dir_output_class" value="src\java\"/>
  <add key="ext_class" value=".java"/>
  <add key="dir_profiles" value="profiles"/>
</appSettings>
```

The *FrameWeb Editor* was built on top of *Eclipse Modeling Tools*,[7] a set of plug-ins for the Eclipse platform (which is Java-based) that offer MDD tools such as meta-modeling, model transformation, verification, etc. FrameWeb meta-models were specified in the Eclipse Modeling Framework (EMF) [5] and the editor developed based on Sirius [14], which offers support for the definition of graphical representations for meta-model elements and provides several features that are common in graphical editors.

The interested reader can obtain detailed instructions on how to install and use the *FrameWeb Editor* at the project's repository.[8]

3.2 The FrameWeb Code Generator

A code generator was created to perform model transformation into code. This process is responsible for creating actual files (artifacts of code) for a WIS modeled in the *FrameWeb Editor*. In this process, we have as input the `.frameweb` file produced by the editor and as output source code files for the system.

The code generator was built in CSharp (in the .NET platform) due to pragmatic reasons (being the language in which the author of this tool is most skilled). However, our short-term future plans include rewriting it in Java in order to integrate it to the *FrameWeb Editor*. It is worth mentioning that the language in which the code generator is built in does not prevent it from generating code in other programming languages, which depend solely on the choice of platform for the WIS being developed (and the existence of *Framework Definitions* for such platform in FrameWeb).

In order to create a flexible tool that can generate code for different languages and frameworks, a few configuration keys were defined, which allow the user to set the paths (folders) in which necessary files for the proper functioning of the generator can be found. This allows one to include support for new platforms and frameworks more easily. Such configuration should be done in the `App.Config` file, a standard configuration file for console-based .NET applications, illustrated in Listing 2. The meaning of each configuration key is explained in Table 1.

As explained in Subsection 3.1, *Framework Definitions* can be imported into a project in *FrameWeb Editor*, allowing developers to incorporate in their models components that are specific for a given framework or platform. Being instances of the *Framework Meta-model*, which is part of FrameWeb's meta-model, such definitions are also saved in `.frameweb` XML files, such as the

[7] http://www.eclipse.org/modeling/tools.php
[8] https://github.com/nemo-ufes/FrameWeb.

Table 1: Description of available configuration keys.

Key	Description
`dir_template`	Root folder in which to find project and language templates.
`project`	The base project, which will be copied to the output folder at the beginning of the code generation process.
`lang`	Folder in which are stored files that are specific to the chosen language, which match the meta-classes from FrameWeb's meta-models.
`dir_output_web`	Subfolder of the project's folder in which to generate the Web-based user interface (e.g., Web pages).
`dir_output_class`	Subfolder of the project's folder in which artifacts of the chosen programming language are generated (e.g., classes).
`ext_class`	File extension for the chosen programming language.
`dir_profiles`	Folder in which *Framework Definition* files can be found.

Listing 3: Fragment of the `MVC_JSF.frameweb` file, which contains the definition (profile) of the JSF *framework*.

```
<packagedElement xsi:type="frameweb:TagLib" name="
    ↪ JSFhtml" URI="http://java.sun.com/jsf/html"
    ↪ prefix="h">
<packagedElement xsi:type="frameweb:Tag"
    ↪ codeGenerationTemplate="inputText.txt" name="
    ↪ inputText"/>
<packagedElement xsi:type="frameweb:Tag"
    ↪ codeGenerationTemplate="commandButton.txt"
    ↪ name="commandButton"/>
```

one shown in Listing 3 for our JSF-based running example. The `MVC_JSF.frameweb` file defines the HTML tags used by this particular Front Controller framework, allowing such tags to be referenced in *Navigation Models* built using the tool. Listing 3 shows two JSF components, `inputText` and `commandButton`, which were used as types for the attributes of `loginForm` in Figure 3.

As we can see in Listing 3, there is an attribute named `codeGenerationTemplate` for each instance of `Tag`, which indicates the name of the template file that represents that particular component (which can be found in the folder specified in the `dir_template` key in `App.Config`). For instance, the template contents for the `inputText` tag is as follows:

```
<h:inputText id="FW_ID" value="#{FW_VALUE}" />
```

These templates include variables which are replaced by the code generator during the model transformation process, such as `FW_ID` and `FW_VALUE` for the `inputText` tag, above. These variables are replaced by values that are read from the `.frameweb` files which

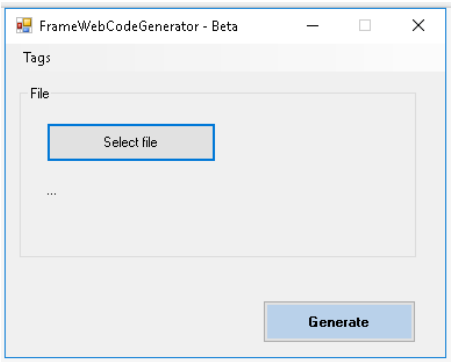

Figure 5: A simple user interface for the code generator.

represent FrameWeb models. Take, for instance, the `login.user` attribute from the `loginForm` class in Figure 3, which is defined as being of type `inputText`. The result of the transformation process for this particular model element would be as follows:

```
<h:inputText id="login_user"
        value="#{loginController.login.user}" />
```

In this example, it is possible to observe that the values obtained from the model of Figure 3 replaced the variables `FW_ID` and `FW_VALUE` in the template, after being adapted to the format of each attribute.

When executed, the code generator loads the meta-model files for the platform and frameworks, which are found in the folder determined by the `dir_profiles` key in the `App.Config` file. This way, support for other platforms and frameworks can be added to the generator in a straightforward manner, by creating a new *Framework Definition* file (also in `.frameweb` format) and the corresponding template files.

Table 2 lists the main variables used by the code generator and their respective meta-classes. When an instance of a given meta-class is found in the FrameWeb model, the generator processes the appropriate template, replacing the variables listed in the table for the given meta-class with the corresponding value used in the model.

While our running example focuses on the *Navigation Model*, the code generator also supports *Entity* and *Application* models, creating source code files for each class in these models. The generation process is analogous to the one described in this subsection, with templates containing the skeleton code for each different class, according to the chosen frameworks and platforms. Support for the *Peristence Model* is currently under development.

To facilitate the use of the code generator, a very simple desktop application was created, as shown in Figure 5. By clicking in its buttons, the user can specify the `.frameweb` file created with the *FrameWeb Editor* and generate code for its models.

3.3 Evolution of the FrameWeb Meta-model

During the development of the *FrameWeb Editor* and the code generator, the original FrameWeb meta-models proposed in [6] had to go through a few corrections and improvements to suit the needs of these tools, namely:

(1) The new attribute `version` was added to meta-class `FrameworkProfile`, in order to account for the version of the *framework* being defined;
(2) The meta-class `Result` was removed and replaced by a new attribute of type `String` in the meta-class `ResultConstraint`, as the value set to the result is always a simple string;
(3) The meta-class `DomainAttribute` was changed from abstract to concrete to allow the creation of domain attributes without any special object/relational mapping (i.e., they assume the default mapping);

Table 2: Main variables and respective meta-classes.

Meta-class: **FrontControllerClass**	
Template variable	Value captured from the model
FW_CLASS_NAME	Name of the class.
FW_BEAN_NAME	Name of the bean, i.e., name used to refer to the class in Web pages.
FW_FRONT_CONTROLLER_PARAMETERS	Attributes of the controller class.
FW_FRONT_CONTROLLER_METHOD	Methods of the controller class.
Meta-class: FrontControllerMethod	
Template variable	Value captured from the model
FW_METHOD_RETURN_TYPE	Return type of the method.
FW_METHOD_NAME	Name of the method.
Meta-class: IOParameter	
Template variable	Value captured from the model
FW_PARAMETER_TYPE	Type fo the parameter.
FW_PARAMETER	Name of the parameter.
FW_PARAMETER_FIRST_UPPER	Same as above, but with first letter in upper case.
Meta-class: UIComponent	
Template variable	Value captured from the model
FW_BODY	Entire contents of the generated page.
FW_ID	Name of the component.
FW_VALUE	Name of the controller and name of the component, separated by a dot.
Meta-class: ServiceInterface	
Template variable	Value captured from the model
FW_INTERFACE_NAME	Name of interface.
FW_INTERFACE_METHOD	Public methods of the interface, captured automatically from the classes taht implement it.

Table 3: Comparison between a real project (course assignment) and generated code from our tool.

Files (xhtml)	Original	Generated	Coverage (%)
formNovoUsuario	68/162	52/66	76% / 40%
formUsuario	69/154	55/66	79% / 42%
listUsuario	32/77	23/31	71% / 40%

Table 4: Comparison between a real project (running example) and generated code from our tool.

Files (xhtml)	Original	Generated	Coverage (%)
index	6/7	6/7	100% / 100%
success	1/1	1/1	100% / 100%

(4) The new attribute *codeGenerationTemplate* was added to the meta-class Tag, in order to specify the name of the template file for each tag of the Front Controller framework;
(5) The attributes *collection*, *fetch*, *order* e *cascade* were added to meta-class DomainConstraints in order to represent Object/Relational mappings that were missing;
(6) The attribute *methodType* was added to meta-classes ServiceMethod, FrontControllerMethod, DAOMethod and DomainMethod in order to represent these method's return types.

4 EVALUATION

The main purpose of FrameWeb's code generator is to decrease the manual effort during the development of Web-based information systems (WISs). Therefore, to evaluate our proposal, we used a course assignment developed by graduate students from our university in the context of a Web Development course,[9] regenerating their code based on FrameWeb models and comparing with the original, to see how much coding effort can be automated.

As done with our running example, we also focused here on the *Navigation Model* for the evaluation, adopting the following methodology: (1) the artifacts related to a given functionality of the system were identified; (2) based on the code written by the students, FrameWeb models were created using the *FrameWeb Editor*; (3) code was generated from the models; (4) the results of the code generation process were compared with the original code by the students. The metrics for evaluation were based on the amount of tags and tag attributes generated versus the actual number of tags (and their attributes) written by the students. Tags that are used for particular purposes (such as
 e <hr />, for page layout) were not included in the count.

Table 3 shows the result of the evaluation using one of the course assignments. The *Original* column displays the amount of tags/attributes found in the original files. The *Generated* column contains the amount of tags/attributes generated automatically by FrameWeb's code generator. The last column shows the percentage of coverage of generated tags/attributes, compared to the original files.

We can observe in Table 3 results above 70% for all generated Web pages with respect to tags, with around 40% of their attributes included, with no effort from the programmer. The low count for attributes was expected, given that FrameWeb models do not include the same level of details as the Web pages themselves. This evaluation, however, can be used as input for further evolutions of the method's meta-models, allowing for new tag attributes to be specified using the *FrameWeb Editor* and, thus, generated automatically.

It is also important to note that some of the course assignments produced by students used container tags, such as panelGroup or panelGrid, which are not yet supported by the code generator (only forms are containers of components). In this sense, the Web Development course is used as an experimentation lab for FrameWeb-related proposals and adding support for container components within forms (multiple-level containment) is on our plans for future work.

The same evaluation was conducted with the running example presented along this paper (cf. Figure 3), the results of which are shown in Table 4. Being a simpler example, all tags and attributes were successfully generated by our tool.

Similar results can also be obtained using other frameworks, for the Java platform or even for a different programming language, as long as they belong to the same framework category (in the case reported here, a front controller framework). For instance, if the above scenarios were to be implemented using CSharp classes as controllers and ASP.NET WebForms for the view, this would be perfectly possible, as long as templates were provided for each component (tags, classes, etc.) in this new platform.

Still, further evaluation efforts using different frameworks/platforms and more examples from course assignments implemented by students are in our plans for future work, in order to evaluate the code generator more thoroughly and identify opportunities for further evolutions of the FrameWeb meta-models. We also intend to evaluate other types of FrameWeb models (*Entity*, *Persistence*, *Application*).

5 RELATED WORK

As mentioned in Section 1, the code generator is an important piece in the evolution of the FrameWeb approach. Previous work in the context of FrameWeb [6, 10, 12], have not proposed any form of automatic model transformation to code, a task which had to be conducted by programmers by interpreting the models and manually writing the code. This paper fills this gap in the FrameWeb method. Below, we discuss related work which also perform automatic code generation, comparing them to our proposal.

MERLIN [7] is an approach focused on the generation of CRUD interfaces (in which users can Create, Retrieve, Update and Delete data from a given entity of the system). The main features of the approach are: (i) generation of screens at runtime; (ii) configuration guided by assisted textual edition of models; (iii) foundations build from standards and languages used by industry; (iv) transparent and managed configuration reuse; and (v) modeling centered on the application's Domain Model. Although similar, FrameWeb is

[9]The source code of the assignment is available at https://github.com/dwws-ufes/2015-searchpapers.

focused on WIS that use frameworks in their architecture and allow modeling of any kind of feature, not only CRUDs. Moreover, MERLIN produces a running system, whereas FrameWeb's code generator outputs source code, which can then be further customized by developers, enhancing it as needed.

MVCASE [9] is an Object-Oriented approach for the development of software that use the Catalysis method and generates code for a distributed platform, based on Enterprise Java Beans (EJB) technology. By using Object-Oriented Programming, MVCASE allows models to be reused, just like the *FrameWeb Editor*. The main advantage of FrameWeb and its meta-model over MVCASE is its extensibility, allowing code to be generated in different frameworks and platforms, as long as they are previously defined and imported into the model.

GeCA [13] allows developers to perform reverse engineering and facilitates system maintenance, as models can be manipulated and the system generated again. Although FrameWeb does not currently support reverse engineering, it does offer support for code generation from its models and based on template files, which allows developers to customize these templates to their needs.

GenERTiCA [15] is a code generation tool that uses UML models together with mapping rules (described in XML). Its goals include the generation of source code as complete as possible (not only skeletons of code, as most code generation tools). This work focuses on automatic code generation for distributed embedded real-time systems, focusing on functional and non-functional requirements and using concepts from Aspect-Oriented Programming. FrameWeb does not currently consider requirements directly in its models and focuses on WIS whose architectures are based on frameworks.

WebML [2] is a modeling notation to visually specify complex websites at the conceptual level, including data input and operation units. WebML is also based in XML and allows the automatic code generation of HTML. In comparison with FrameWeb, its user interfaces are more precise, since its models provide details on the organization of components in Web pages, differently from FrameWeb, which only models which are the components of a page, but not their arrangement. However, FrameWeb models the entire WIS, not only Web pages, allowing the generation of code for all system tiers (from *Presentation* to *Data Access*).

6 CONCLUSIONS

In this paper, we propose a solution for code generation based on the FrameWeb method and language, which includes a graphical editor for creating the models and evolutions to the original meta-models in order to support both tools. We believe this contribution can motivate the development of WISs based on models by guiding the steps of the development and relieving programmers from much of the coding effort.

The code generator is available for use and show promising results in its initial evaluations. Nonetheless, this is a work in progress. Future work includes its seamless integration with the *FrameWeb Editor*; the evolution of the meta-model and code generator in order to support the generation of CRUD features; and support for FrameWeb-LD [10], an extension that models linked data mappings in FrameWeb *Entity Models*.

We also intend to further evaluate FrameWeb related tools by having students from the Web Development course at our university use them in the context of their course assignments, gathering feedback from practitioners. In the long term, we also intend to evaluate the use of software product lines and generative programming techniques [3] in order to generate families of applications in the same problem domain.

Finally, as the FrameWeb method itself evolves, the code generator should evolve accordingly, with added support to new types of frameworks (e.g., authentication frameworks) or platforms (mobile devices, use of Web APIs, etc.).

ACKNOWLEDGMENTS

NEMO (http://nemo.inf.ufes.br) is currently supported by Brazilian research agencies FAPES (# 0969/2015), CNPq (# 485368/2013-7, # 461777/2014-2), and by UFES' FAP (# 6166/2015).

REFERENCES

[1] Deepak Alur, John Crupi, and Dan Malks. 2003. *Core J2EE Patterns: Best Practices and Design Strategies* (2nd ed.). Prentice Hall / Sun Microsystems Press.
[2] A. Bongio, S. Ceri, P. Fraternali, and Mauhino. 2000. Web Modeling Language (WebML): a modeling language for designing Web sites. (2000).
[3] Krzysztof Czarnecki, Kasper Østerbye, and Markus Völter. 2002. Generative Programming. In *Proceedings of the Workshops and Posters on Object-Oriented Technology (ECOOP '02)*. Springer-Verlag, 15–29.
[4] William B. Frakes and Kyo Kang. 2005. Software reuse research: Status and future. *IEEE Transactions on Software Engineering* 31, 7 (2005), 529–536.
[5] Richard C. Gronback. 2009. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit* (1 ed.). Addison-Wesley Professional.
[6] Beatriz Franco Martins and Vítor E. Silva Souza. 2015. A Model-Driven Approach for the Design of Web Information Systems based on Frameworks. In *Proc. of the 21st Brazilian Symposium on Multimedia and the Web*. ACM, 41–48.
[7] Marcelo Mrack. 2009. *Geração Automática e Assistida de Interfaces de Usuário*. Technical Report. Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação, Universidade Federal Do Rio Grande Do Sul.
[8] Oscar Pastor, Sergio España, José Ignacio Panach, and Nathalie Aquino. 2008. Model-driven development. *Informatik-Spektrum* 31 (2008), 394–407.
[9] Antônio Francisco Prado and Daniel Lucrédio. 2001. Ferramenta MVCASE - Estágio Atual: Especificação, Projeto e Construção de Componentes. (2001).
[10] *removed for doubleblind review*. 2016. A Framework-based Approach for the Integration of Web-based Information Systems on the Semantic Web. In *Proc. of the 22nd Brazilian Symposium on Multimedia and the Web*. ACM, 231–238.
[11] Ian Sommerville. 2010. *Software Engineering* (9th ed.). Addison-Wesley Publishing Company, USA.
[12] Vítor E. S. Souza, Ricardo A. Falbo, and Giancarlo Guizzardi. 2009. Designing Web Information Systems for a Framework-based Construction. In *Innovations in Information Systems Modeling: Methods and Best Practices* (1 ed.), Terry Halpin, Eric Proper, and John Krogstie (Eds.). IGI Global, Chapter 11, 203–237.
[13] Igor Steinmacher, Éderson Fernando Amorim, Flávio Luiz Schiavoni, and Elisa Hatsue Moriya Huzita. 2006. GeCA: Uma Ferramenta de Engenharia Reversa e Geração Automática de Código. (2006).
[14] Vladimir Viyovic, Milan Maksimovic, and Branko Perisic. 2014. Sirius: A rapid development of DSM graphical editor. In *Intelligent Engineering Systems (INES), 2014 18th International Conference on*. IEEE, 233–238.
[15] Marco A. Wehrmeister, Edison P. Freitas, Carlos E. Pereira, and Franz Rammig. 2008. GenERTiCA: A Tool for Code Generation and Aspects Weaving. (2008).