Integração de um Gerador de Código ao FrameWeb Editor

Breno Leite Zupeli
Núcleo de Estudos em Modelagem Conceitual e Ontologias (NEMO), Departamento de Informática, Universidade Federal do Espírito Santo (UFES) - Vitória, ES, Brasil
brenolzupeli@gmail.com

Vítor E. Silva Souza
Núcleo de Estudos em Modelagem Conceitual e Ontologias (NEMO), Departamento de Informática, Universidade Federal do Espírito Santo (UFES) - Vitória, ES, Brasil
vitor.souza@ufes.br

ABSTRACT

FrameWeb (Framework-based Design Method for Web Engineering) incorporates concepts from categories of frameworks commonly used in the development of Web-based Information Systems into design models, defining the syntax of such models with meta-models. Based on Model-Driven Development (MDD) techniques, a CASE tool called *FrameWeb Editor* was built. In a separate effort, a code generation tool was proposed, but did not use the method's MDD foundations. In this paper, we report on the integration of the code generator into the FrameWeb Editor and the FrameWeb meta-model.

KEYWORDS

Web Engineering, Frameworks, Model-Driven Development, FrameWeb, CASE Tool

1 INTRODUÇÃO

No contexto da Engenharia Web, é muito comum o reuso de código de infraestrutura por meio do uso de *frameworks* (ex.: Spring Framework[1]) ou plataformas de desenvolvimento (ex.: Java EE[2]). Um *framework* é um artefato de código que provê componentes prontos que podem ser reutilizados mediante configuração, composição ou herança [18]. Seu uso auxilia a evitar a contínua redescoberta e reinvenção de padrões e componentes arquiteturais básicos, reduzindo custos e melhorando a qualidade do software, devido à reutilização de arquiteturas bem estabelecidas e testadas [17].

Com esta motivação, o método *FrameWeb* [18, 19] (*Framework-based Design Method for Web Engineering*) define uma arquitetura padrão para facilitar a integração com tais *frameworks*, além de propor um conjunto de modelos que traz para o projeto arquitetural do sistema conceitos inerentes a eles. O método é baseado em conceitos de MDD (*Model-Driven Development*) [16], sendo a sintaxe de sua linguagem de modelagem especificada formalmente por meio de meta-modelos [12], o que permite que suporte a novos *frameworks* sejam incluídos e os modelos de projeto sejam adaptados ao conjunto de *frameworks* que se deseja utilizar.

A abordagem MDD promove, também, a construção de ferramentas de apoio ao uso do método. Campos & Souza propõem o *FrameWeb Editor* [5], uma ferramenta CASE que dá suporte à criação e verificação dos modelos propostos por FrameWeb de forma gráfica e baseada nos meta-modelos que definem a sintaxe de sua linguagem. O editor foi construído com base no projeto Sirius [20], que por sua vez é baseado no *Eclipse Modeling Framework* (EMF) [9], que permite o desenvolvimento de ferramentas dentro da abordagem MDD na plataforma Eclipse.[3]

Em um esforço paralelo, Almeida et al. [7] propõem um gerador capaz de criar esqueletos de código a partir dos modelos prescritos por FrameWeb, adaptados ao conjunto de *frameworks* escolhido para o projeto em questão. No entanto, o gerador de código não foi desenvolvido dentro da abordagem MDD, i.e., não utiliza o meta-modelo para especificar as extensões da linguagem necessárias para a geração de código (ex.: *templates* de código utilizados pelo gerador), nem faz uso de técnicas e ferramentas de transformação oferecidas pela abordagem MDD e pela plataforma Eclipse, tendo sido desenvolvido na plataforma .NET como uma ferramenta separada do FrameWeb Editor.

O objetivo deste artigo é apresentar os resultados de um esforço de integração do gerador de código [7] ao FrameWeb Editor [5], detalhando as modificações necessárias nos meta-modelos de FrameWeb para a adequação do gerador à abordagem MDD. A Seção 2 introduz o referencial teórico, a Seção 3 apresenta as mudanças feitas no FrameWeb Editor e nos meta-modelos de FrameWeb para integração do gerador de código, a Seção 4 compara trabalhos relacionados e a Seção 5 traz as considerações finais.

2 REFERENCIAL TEÓRICO

Para auxiliar na construção de Sistemas de Informação Web (*Web-based Information Systems* ou WIS) que possuam uma infraestrutura arquitetônica baseada no uso de *frameworks*, o método FrameWeb (*Framework-based Design Method for Web Engineering*) [18, 19] propõe uma arquitetura de software padrão, que divide o sistema em três grandes camadas — lógica de apresentação, lógica de negócio e lógica de acesso aos dados — seguindo o padrão arquitetônico *Service Layer* (Camada de Serviço) [8].

Com base nessa arquitetura, FrameWeb define quatro tipos de diagramas, todos baseados no Diagrama de Classes da UML [15], que são utilizados para representar os componentes específicos de cada camada em questão, abrangendo os elementos típicos da plataforma Web e relacionados aos *frameworks* utilizados.

Na camada de apresentação, o pacote View (visão) contém os elementos de interface gráfica com o usuário para a plataforma Web, como páginas HTML, imagens, folhas de estilo, scripts JavaScript, etc. O pacote Controller (controle) engloba as classes de controle, responsáveis por capturar os estímulos e os dados enviados pelo usuário por meio dos componentes de visão, chamar os serviços apropriados na aplicação e apresentar o resultado de volta ao usuário. Tal arquitetura é baseada nos *frameworks* Web

[1] https://spring.io.
[2] http://www.oracle.com/technetwork/java/javaee/overview/index.html.
[3] http://www.eclipse.org.

In: XVII Workshop de Ferramentas e Aplicações (WFA 2018), Salvador, Brasil. Anais do XXIV Simpósio Brasileiro de Sistemas Multimídia e Web: Workshops e Pôsteres. Porto Alegre: Sociedade Brasileira de Computação, 2018.

ISBN 978-85-7669-435-9.

(ou controlador frontal, ex.: *JavaServer Faces*[4]) e representada no **Modelo de Navegação**.

Na camada de negócio, o pacote Domain (domínio) contém as classes que representam os elementos do domínio do problema (conceitos do mundo real), cuja persistência é realizada por *frameworks* de mapeamento objeto/relacional (ex.: *Java Persistence API*[5]). Tais classes e seus mapeamentos são representados no **Modelo de Entidades**. Já o pacote Application (aplicação) reúne as classes de serviço, responsáveis pela implementação dos casos de uso (funcionalidades do WIS), que são representadas no **Modelo de Aplicação**, juntamente com suas dependências, gerenciadas pelo *framework* de injeção de dependências (ex.: *Contexts and Dependency Injection for Java*[6]).

Por fim, na camada de acesso a dados, o pacote Persistence (persistência) traz classes responsáveis pela persistência dos objetos de domínio, segundo o padrão de projeto *Data Access Object* (DAO) [1]. Os DAOs são representados no **Modelo de Persistência**.

Por serem baseados no Diagrama de Classes da UML, qualquer editor UML pode ser utilizado para construir os modelos descritos acima. No entanto, para garantir que os modelos sejam construídos utilizando apenas construtos válidos para o FrameWeb, a sintaxe da *linguagem FrameWeb* foi especificada formalmente por meio de meta-modelos [12], seguindo a abordagem MDD [16].

O Desenvolvimento Orientado a Modelos (*Model-Driven Development*, ou MDD) propõe uma abordagem na qual os modeladores concentram esforços na elaboração de modelos conceituais para representação de todas as características de um sistema, e possibilita modelar as funcionalidades e recursos desejáveis para o sistema, portanto, o modelo gerado é o projeto do sistema em si. Posteriormente, por meio de transformações entre modelos de diversos níveis de abstração é feita a implementação do sistema. Algumas das principais vantagens do MDD para o processo de geração de código são: **portabilidade**, porque os modelos de alto nível podem ser transformados em códigos diferentes, dependendo da plataforma ou *framework* escolhidos; **corretude**, pois geradores de código não produzem erros acidentais e além disso facilitam a identificação de erros conceituais, visto que ocorrem em níveis mais altos de abstração; e **otimização**, dado que os modelos possibilitam mais eficiência e menor incidência de erros aos geradores.

A partir do meta-modelo de FrameWeb, foi construído o *FrameWeb Editor*, que oferece um ambiente gráfico que possibilita a elaboração destes diagramas, bem como a definição da plataforma de programação e dos *frameworks* utilizados. A Figura 1 apresenta uma visão geral do editor, que possui do lado esquerdo o *Project Explorer* para visualização do projeto, no lado direito apresenta um painel com as opções para criação de componentes de acordo com o modelo utilizado, e na parte de baixo são mostradas as opções para configuração do atributo selecionado. Um duplo-clique em um dos componentes de modelo permite ao desenvolvedor construir o modelo em questão. A figura mostra, também, que para este projeto foi escolhida a plataforma Java e o *framework* JSF.

A fim de facilitar o desenvolvimento de projetos elaborados no *FrameWeb Editor*, um gerador de código foi desenvolvido para geração do esqueleto do código, baseando-se no arquivo gerado

[4]JSF, http://jcp.org/en/jsr/detail?id=344
[5]JPA, http://jcp.org/en/jsr/detail?id=338
[6]CDI, http://jcp.org/en/jsr/detail?id=346

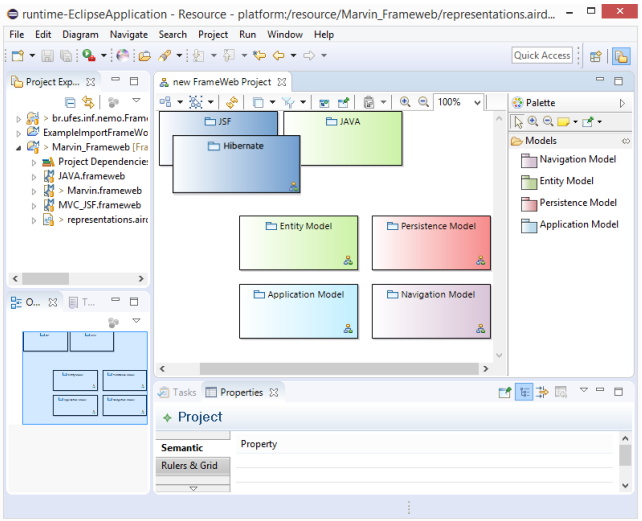

Figura 1: Visão geral do FrameWeb Editor [5].

pelo editor com as informações dos modelos criados, visto que este gerador não se encontrava integrado ao *FrameWeb Editor*. A próxima sessão apresenta como foi feita essa integração.

3 INTEGRAÇÃO DO GERADOR DE CÓDIGO

Partiu-se do FrameWeb Editor [5] e do gerador de código [7], reimplementando este último usando a linguagem Java (sua implementação original foi feita em C#), para possibilitar sua integração ao meta-modelo do FrameWeb e, assim, ao editor.

O gerador de código em sua versão original armazena os *templates* por meio de arquivos organizados em diretórios, um arquivo de configuração deve ser modificado para indicar os diretórios dos *templates* necessários e o diretório onde o código gerado será armazenado, isso não permite uma boa usabilidade ao usuário, além de não promover uma ligação explícita destes *templates* com a linguagem FrameWeb, que é definida por meta-modelos.

A fim de permitir a utilização de vários tipos de *frameworks* e de integrar os *templates* ao meta-modelo do FrameWeb, foram adicionadas três novas meta-classes, representadas na Figura 2: **FrontControllerTemplate**, responsável por armazenar *templates* das páginas, dos componentes das páginas, das classes controladoras, seus métodos e atributos; **DITemplate**, responsável por armazenar *templates* das classes de serviço, seus métodos e atributos; e **ORMTemplate**, responsável por armazenar *templates* das classes de domínio e seus atributos. Além disso, a classe **Tag**, já existente, ganhou o atributo codeGenerationTemplate para armazenamento dos *templates* das *tags* HTML utilizadas nos Modelos de Navegação.

Instâncias destas novas meta-classes podem ser vistas na Figura 1, apresentada anteriormente. O elemento JSF representa uma instância da classe FrontControllerTemplate e armazena os *templates* das classes, métodos, páginas, etc. Já Hibernate é instância da classe ORMTemplate e armazena os *templates* de classes e atributos. Estes *frameworks* são utilizados nos modelos de Navegação e Entidades, respectivamente.

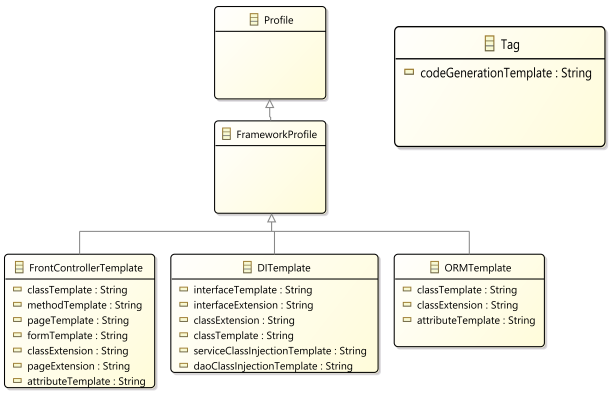

Figura 2: Alterações feitas no meta-modelo de FrameWeb.

Devido ao fato de possuírem caracteres especiais, tais *templates* são transformados por meio de um codificador de URLs (já disponível na API da linguagem Java) para armazenamento nos atributos das respectivas meta-classes, sendo decodificados em tempo de execução pelo gerador de código. Os *templates* utilizam variáveis que possuem o prefixo FW_, cada uma delas correspondendo a um atributo de uma meta-classe do meta-modelo de FrameWeb, ou seja, referente a alguma informação que será extraída dos modelos criados com o FrameWeb Editor, conforme já apresentado em [7].

A Figura 3 mostra um exemplo de Modelo de Navegação, representando uma funcionalidade do Sistema de Controle de Afastamento de Professores (SCAP), uma aplicação que auxilia departamentos de universidades federais a gerenciar pedidos de afastamento (*leave of absence requests*) de seus professores. O modelo traz uma página chamada form.xhtml, que possui um formulário a ser preenchido pelo professor que deseja solicitar seu afastamento, cujos dados são enviados à classe controladora RequestLeaveController, que tem o seu método submit() associado a esta submissão. Ao final do registro da solicitação de afastamento, o controlador redireciona o usuário para a página success.xhtml.

Com a integração do gerador de código ao FrameWeb Editor, para ativar a geração de código, o usuário deve, na visão geral do projeto no editor (Figura 1), clicar com o botão direito e selecionar a opção *Gerar código*. Uma janela é exibida para a seleção do diretório desejado para a criação do projeto com o código gerado. Na sequência, detalhamos como esta geração de código acontece utilizando o exemplo da Figura 3.

Para que as páginas HTML e seus formulários («page» e «form» no Modelo de Navegação) sejam gerados a partir do modelo, na definição do *framework* JSF (Figura 1), como instância da meta-classe FrontControllerTemplate (Figura 2), os atributos pageTemplate e formTemplate foram preenchidos, respectivamente, com os conteúdos das listagens 1 e 2. Na página Web, a variável FW_BODY é substituída por formulários que compõem a página, enquanto que esta mesma variável no *template* do formulário é substituída pelos seus componentes visuais (representados por seus atributos no Modelo de Navegação).

Também na definição do JSF, as diferentes *tags* que representam os componentes visuais utilizados em formulários Web providas

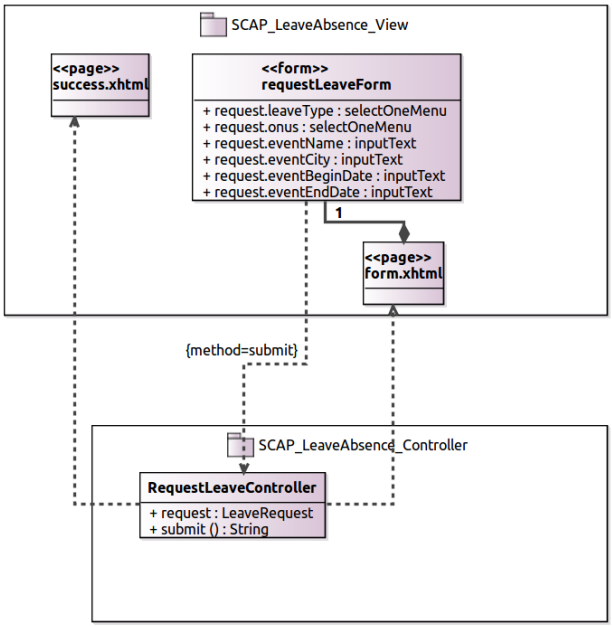

Figura 3: Exemplo de um modelo de Nagevação.

Listagem 1: *Template* para páginas HTML (pageTemplate).

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.
    ↪ w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:f="http://java.sun.com/jsf/core"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:ui="http://java.sun.com/jsf/facelets">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    ↪
<title>Insert title here</title>
</head>
<h:body>
FW_BODY
</h:body>
</html>
```

Listagem 2: *Template* para form HTML (formTemplate).

```
<h:form id="FW_ID">
FW_BODY
</h:form>
```

Listagem 3: *Template* para campo JSF inputText.

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

pelo *framework* são registradas como instâncias da meta-classe Tag e também associadas a um *template* de código. A Listagem 3 mostra o *template* para componentes de texto simples (<h:inputText />) enquanto a Listagem 4 se refere a componentes de caixa de seleção (<h:selectOneMenu />). Neste caso, a variável FW_VALUE é substituída pelo nome do componente (atributo da classe «form» no Modelo de Navegação) para que o JSF faça a ligação (*binding*) entre o componente Web e o controlador Java.

Listagem 4: *Template* para campo JSF selectOneMenu.

```
<h:selectOneMenu id="FW_ID" value="#{FW_VALUE}"></h:selectOneMenu><
      ↪ br />
```

Listagem 5: Parte do arquivo `form.xhtml` gerado.

```
...
<h:body>
<h:form id="requestLeaveForm">
<h:selectOneMenu id="request_leaveType" value="#{requestLeaveForm.
      ↪ request.leaveType}"></h:selectOneMenu><br />
...
<h:inputText id="request_eventEndDate" value="#{requestLeaveForm.
      ↪ request.eventEndDate}" /><br />
</h:form>
</h:body>
</html>
```

Listagem 6: *Template* para controladora (classTemplate).

```
package FW_FRONT_CONTROLLER_PACKAGE;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean(name = "FW_BEAN_NAME") @SessionScoped
public class FW_CLASS_NAME {
       FW_FRONT_CONTROLLER_ATTRIBUTES
       public FW_CLASS_NAME() { }
       FW_FRONT_CONTROLLER_METHODS
}
```

Listagem 7: *Template* para atributo (attributeTemplate).

```
private FW_ATTRIBUTE_TYPE FW_ATTRIBUTE;
public FW_ATTRIBUTE_TYPE getFW_ATTRIBUTE_FIRST_UPPER() {
       return FW_ATTRIBUTE;
}
public void setFW_ATTRIBUTE_FIRST_UPPER(FW_ATTRIBUTE_TYPE
      ↪ _FW_ATTRIBUTE) {
       FW_ATTRIBUTE = _FW_ATTRIBUTE;
}
```

Ao utilizar todos estes *templates* para gerar código do elemento form.xthml do modelo da Figura 3, parte do resultado é representada na Listagem 5.

Analogamente, para que a classe controladora seja gerada, a definição do *framework* JSF traz nos atributos classTemplate, attributeTemplate e methodTemplate, respectivamente, o conteúdo das listagens 6, 7 e 8. O *template* da classe traz as anotações do JSF para que a classe possa ser referenciada a partir das páginas Web, bem como variáveis que são substituídas pelo código gerado a partir dos atributos (FW_FRONT_CONTROLLER_ATTRIBUTES) e métodos (FW_FRONT_CONTROLLER_METHODS).

O *template* de atributos gera o código do atributo em si, juntamente com métodos de acesso (get/set) utilizados pelo JSF. Por fim, o *template* de métodos apenas inclui um esqueleto do método, sem implementação. Combinando tais *templates* na geração da classe RequestLeaveController da Figura 3, o resultado é o exibido pela Listagem 9.

Conforme proposto em [12], o FrameWeb pode ser estendido por meio da criação de novos perfis de *framework*. Com a integração do gerador de código ao FrameWeb Editor, tal esforço envolve, agora, instanciar as classes da Figura 2 e prover os *templates* de código para que o gerador possa construir, a partir dos respectivos modelos,

Listagem 8: *Template* para método (methodTemplate).

```
public FW_METHOD_RETURN_TYPE FW_METHOD_NAME(){
       return null;
}
```

Listagem 9: Classe RequestLeaveController gerada.

```
package br.ufes.inf.scap.control;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean(name = "requestLeaveController") @SessionScoped
public class RequestLeaveController {
       private LeaveRequest request;
       public LeaveRequest getRequest() {
              return request;
       }
       public void setRequest(LeaveRequest _request){
              request = _request;
       }
       public RequestLeaveController() {}
       public String submit() {
              return null;
       }
}
```

os artefatos de código de maneira adequada, conforme ilustrado acima para o *framework* JSF. O próprio editor provê suporte para definição de novos *frameworks* em sua interface gráfica [5].

A ferramenta FrameWeb Editor, que possui o gerador de código integrado, encontra-se sob a licença MIT e o leitor interessado pode obter instruções detalhadas de instalação e uso na página do repositório do projeto, hospedada em https://github.com/nemo-ufes/FrameWeb/wiki/A-Ferramenta-FrameWeb-Editor.

4 TRABALHOS RELACIONADOS

Assim como FrameWeb, outros métodos de Engenharia Web possuem ferramentas CASE e/ou geradores de código associados.

O método UWE (UML-based Web Engineering) [10] que, assim como FrameWeb, é baseado em MDD, possui a ferramenta CASE MagicUWE,[7] porém a mesma não oferece suporte a geração de código. Uma outra ferramenta, UWE4JSF,[8] permite gerar aplicações JSF baseada numa versão estendida do perfil UWE, porém não é integrada ao editor. Outro método MDD, o OOH4RIA [13], focado no desenvolvimento de *Rich Internet Applications*, possui a ferramenta associada OIDE [14] (OOH4RIA IDE), que gera código para os *frameworks* Google Web Toolkit e Silverlight.

O RUX-Method [4], focado em criação de interfaces sensíveis a contexto, possui uma ferramenta chamada RUX-Tool [11], capaz de gerar código de componentes de interface com o usuário automaticamente, devendo porém ser usada em conjunto com a ferramenta comercial WebRatio.[9] Esta ferramenta utiliza a linguagem visual IFML [2] para modelagem de fluxos de interação, controle de comportamento e conteúdo de interfaces com o usuário (*front-ends*), linguagem adotada pela OMG[10] como padrão em 2013. Alternativas de código aberto para edição deste tipo de modelo, como o IFMLEdit.org [3] também oferecem geração de código.

[7] http://uwe.pst.ifi.lmu.de/toolMagicUWE.html.
[8] http://uwe.pst.ifi.lmu.de/toolUWE4JSF.html.
[9] https://www.webratio.com.
[10] https://www.omg.org/.

Há geradores de código que trabalham com modelos UML e arquivos de mapeamento (por exemplo, em XML) para a geração de código de aplicações Web. GenERTiCA [21] se propõe a gerar código de forma mais completa possível para sistemas de tempo real distribuídos, utilizando conceitos da Programação Orientada a Aspectos. WebML [6] foca na especificação visual de websites complexos, permitindo a geração de código das páginas HTML que compõem seu *front-end*.

O principal diferencial da abordagem FrameWeb — incluindo seu editor e gerador de código, integrados neste trabalho — em relação a outras abordagens, inclusive ferramentas não-acadêmicas como, por exemplo, JHipster,[11] é a sua extensibilidade. Seus meta-modelos permitem que os usuários do FrameWeb Editor adicionem suporte a novos *frameworks* (inclusive legados, que dificilmente teriam suporte de ferramenta), incluindo *templates* que possibilitam a geração de código para tais *frameworks*. Tais *templates* também podem ser personalizados pelos usuários do FrameWeb Editor.

5 CONCLUSÕES

Neste artigo, apresentamos o resultado de um esforço de integração entre a ferramenta CASE FrameWeb Editor [5] e o gerador de código [7] proposto para o método FrameWeb, que se fez necessário pois este último não foi construído com base no meta-modelo FrameWeb [12].

A integração se faz necessária para que os usuários do método tenham uma interface unificada para aproveitar a principal vantagem de FrameWeb, sua extensibilidade: os próprios usuários finais do FrameWeb Editor podem cadastrar novos *frameworks* e modificar os *templates* para geração de código, adaptando-os às necessidades do projeto ou da organização na qual trabalham. Em particular, projetos legados que utilizam *frameworks* que não possuem suporte de ferramentas comerciais ou *open source* disponíveis podem ser beneficiar desta capacidade de FrameWeb de se adaptar a diferentes conjuntos de *frameworks*.

A integração do editor e do gerador em uma única ferramenta traz como vantagens uma maior facilidade de uso, pois o usuário final não precisa instalar e usar duas ferramentas separadas, de plataformas diferentes; diminuição da ambiguidade em modelos, pois são gerenciados por uma única ferramenta; e uma manutenção mais fácil para os desenvolvedores FrameWeb, que não precisam manter duas bases de código diferentes, em linguagens de programação diferentes. A construção do gerador dentro da plataforma Eclipse permitirá, também, o uso dos recursos de Desenvolvimento Orientado a Modelos presentes na plataforma.

Como trabalhos futuros, pretende-se utilizar o FrameWeb Editor e seu gerador de código integrados na atual edição do curso de *Desenvolvimento Web e Web Semântica*[12] em nossa universidade, avaliando-o junto aos alunos matriculados nesta disciplina. Também encontra-se em andamento, no contexto de um projeto de Iniciação Científica, uma maior integração do FrameWeb Editor com a IDE Eclipse, de modo a aproximar a ferramenta da indústria de desenvolvimento de software, promovendo transferência de tecnologia da pesquisa sobre FrameWeb da academia para as empresas do ramo e permitindo formas mais robustas de avaliação da mesma.

[11] https://jhipster.tech.
[12] https://www.inf.ufes.br/~vitorsouza/teaching/dwws-20182/.

ACKNOWLEDGMENTS

O NEMO (http://nemo.inf.ufes.br) possui atualmente apoio do CNPq (processo 407235/2017-5), da CAPES (23038.028816/2016-41) e da FAPES (69382549/2015). Este trabalho foi realizado com bolsa do Programa Institucional de Iniciação Científica da UFES.

REFERÊNCIAS

[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] Luciano Baresi, Franca Garzotto, and Paolo Paolini. 2001. Extending UML for modeling web applications. In *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on*. IEEE, 10–pp.
[3] Carlo Bernaschina, Sara Comai, and Piero Fraternali. 2017. IFMLEdit.Org: Model Driven Rapid Prototyping of Mobile Apps. In *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft '17)*. IEEE Press, 207–208. https://doi.org/10.1109/MOBILESoft.2017.15
[4] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. 2003. A unifying reference framework for multi-target user interfaces. *Interacting with computers* 15, 3 (2003), 289–308.
[5] Silas Louzada Campos and Vítor E. S. Souza. 2017. FrameWeb Editor: Uma Ferramenta CASE para suporte ao Método FrameWeb. In *Anais do 16º Workshop de Ferramentas e Aplicações, 23º Simpósio Brasileiro de Sistemas Multimedia e Web*. SBC, Gramado, RS, Brazil, 199–203.
[6] Stefano Ceri, Piero Fraternali, and Aldo Bongio. 2000. Web Modeling Language (WebML): a modeling language for designing Web sites. *Computer Networks* 33, 1-6 (jun 2000), 137–157. https://doi.org/10.1016/S1389-1286(00)00040-2
[7] Nilber V. de Almeida, Silas L. Campos, and Vítor E. S. Souza. 2017. A Model-Driven Approach for Code Generation for Web-based Information Systems Built with Frameworks. In *Proc. of the 23rd Brazilian Symposium on Multimedia and the Web*. ACM, Gramado, RS, Brazil, 245–252. https://doi.org/10.1145/3126858.3126863
[8] Martin Fowler. 2002. *Patterns of Enterprise Application Architecture* (1 ed.). Addison-Wesley.
[9] Richard C. Gronback. 2009. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit* (1 ed.). Addison-Wesley Professional.
[10] Nora Koch, Alexander Knapp, Gefei Zhang, and Hubert Baumeister. 2008. Web Engineering: Modelling and Implementing Web Applications. In *UML-Based Web Engineering*. Springer, London, UK.
[11] Marino Linaje, Juan Carlos Preciado, Rober Morales-Chaparro, Roberto Rodríguez-Echeverría, and Fernando Sánchez-Figueroa. 2009. Automatic Generation of RIAs Using RUX-Tool and Webratio. In *Web Engineering: 9th International Conference, ICWE 2009, Proceedings*. Springer Berlin Heidelberg, San Sebastián, Spain, 501–504. https://doi.org/10.1007/978-3-642-02818-2_48
[12] Beatriz Franco Martins and Vítor E. S. 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.
[13] Santiago Meliá, Jaime Gómez, Sandy Pérez, and Oscar Díaz. 2008. A model-driven development for GWT-based rich internet applications with OOH4RIA. In *Web Engineering, 2008. ICWE'08. Eighth International Conference on*. IEEE, 13–23.
[14] Santiago Meliá, Jose-Javier Martínez, Sergio Mira, Juan Antonio Osuna, and Jaime"Gómez. 2010. An Eclipse Plug-in for Model-Driven Development of Rich Internet Applications. In *Web Engineering*. Springer, 514–517.
[15] OMG. 2017. Unified Modeling Language, version 2.5.1, https://www.omg.org/spec/UML/2.5.1/. (2017).
[16] Oscar Pastor, Sergio España, José Ignacio Panach, and Nathalie Aquino. 2008. Model-driven development. *Informatik-Spektrum* 31 (2008), 394–407.
[17] D.C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. 2013. *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects*. Wiley.
[18] Vítor E Silva Souza. 2007. *FrameWeb: um Método baseado em Frameworks para o Projeto de Sistemas de Informação Web*. Master's thesis. Programa de Pós-Graduação em Informática, Universidade Federal do Espírito Santo.
[19] 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.
[20] 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.
[21] Marco A. Wehrmeister, Edison P. Freitas, Carlos E. Pereira, and Franz Rammig. 2008. GenERTiCA: A Tool for Code Generation and Aspects Weaving. In *Proc. of the 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*.