Nilber Vittorazzi de Almeida

Uma Abordagem Orientada a Modelos para Geração de Código para Sistemas de Informação Baseados na Web construídos com Frameworks

Vitória, ES

2018

Nilber Vittorazzi de Almeida

Uma Abordagem Orientada a Modelos para Geração de Código para Sistemas de Informação Baseados na Web construídos com Frameworks

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Programa de Pós-Graduação em Informática

Orientador: Prof. Dr. Vítor E. Silva Souza

Vitória, ES

2018

Nilber Vittorazzi de Almeida Uma Abordagem Orientada a Modelos para Geração de Código para Sistemas de Informação Baseados na Web construídos com Frameworks/ Nilber Vittorazzi de Almeida. – Vitória, ES, 2018-
68 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Vítor E. Silva Souza

Dissertação de Mestrado – Universidade Federal do Espírito Santo – UFES
Centro Tecnológico
Programa de Pós-Graduação em Informática, 2018.

1. Engenharia Web. 2. Frameworks, Desenvolvimento Dirigido por Modelos. 3. Geração de Código. 4. FrameWeb.

CDU 02:141:005.7

Nilber Vittorazzi de Almeida

Uma Abordagem Orientada a Modelos para Geração de Código para Sistemas de Informação Baseados na Web construídos com Frameworks

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

Prof. Dr. Vítor E. Silva Souza
Orientador

Tayana Uchoa Conte
Universidade Federal do Amazonas (UFAM)

Ricardo de Almeida Falbo
Universidade Federal do Espírito Santo (UFES)

Vitória, ES
2018

Agradecimentos

A Deus, por ter conduzido meus passos até agora.
Ao meu orientador, professor Vítor Estêvão Silva Souza, pela oportunidade concedida, pela orientação, dedicação e conhecimento repassado. Os meus sinceros agradecimentos.
Aos meus colegas e ex-colegas de mestrado.
A todos aqueles que direta ou indiretamente contribuíram para a realização deste trabalho.

"Se A é o sucesso, então A é igual a X mais Y mais Z. O trabalho é X; Y é o lazer; e Z é manter a boca fechada..

(Albert Einstein)

Resumo

No campo da Engenharia Web, existem diversos métodos propostos para o desenvolvimento de sistemas de informação para a Web. *FrameWeb* é um método que se concentra em sistemas que utilizam certos tipos de *frameworks* em sua arquitetura, propondo o uso de modelos que incorporam conceitos desses *frameworks* durante o projeto. Tais conceitos estão representados nos construtos da linguagem, cuja sintaxe (abstrata) é definida por um metamodelo, permitindo a extensão do método. Este trabalho propõe uma ferramenta de geração de código a partir dos modelos FrameWeb, baseada numa extensão do metamodelo da linguagem.

O *Gerador FrameWeb*, artefato alvo deste trabalho, é capaz de transformar um modelo criado previamente com base no metamodelo FrameWeb — utilizando a ferramenta CASE *FrameWeb Editor* — num projeto de software estruturalmente completo para ser usado como base no desenvolvimento de um projeto Web. Ao final do processo é possível observar as vantagens da utilização do FrameWeb, pois não somente as classes e suas estruturas básicas são geradas, mas também a parte visual do projeto elaborada no editor FrameWeb, sendo esse um dos grandes diferenciais propostos pelo método: a representação de uma página e seus componentes diretamente no modelo.

O Gerador FrameWeb foi avaliado por meio de experimentos em laboratório, nos quais projetos previamente desenvolvidos por alunos de uma disciplina de Desenvolvimento Web tiveram seus modelos FrameWeb construídos no editor e o código gerado foi comparado com o original para que a cobertura da geração de código pudesse ser medida. Tais experimentos mostraram que um alto percentual do código pôde ser gerado automaticamente, liberando os desenvolvedores da parte mais tediosa da codificação.

Palavras-chaves: Engenharia Web, Frameworks, Desenvolvimento Orientado a Modelos, Geração de Código, FrameWeb

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 the use of models that incorporate concepts of such frameworks at design-time. Such concepts are represented in the language constructs, whose (abstract) syntax is defined by a metamodel, allowing for the method to be extended. This work proposes a code generation tool from FrameWeb models based on an extension of the language's metamodel.

The FrameWeb Generator, target artifact of this work, is able to transform a previously created model based on the FrameWeb metamodel — using the CASE tool *FrameWeb Editor* — in a structurally complete software project to be used as basis in the development of a Web project. At the end of the process it is possible to observe the advantages of using FrameWeb, since not only the classes and their basic structures are generated, but also the visual part of the project elaborated in the FrameWeb editor, being one of the great features proposed by the method: the representation of a page and its components directly in the model.

The FrameWeb Generator was evaluated through laboratory experiments in which projects previously developed by students of a Web Development course had their FrameWeb models built in the editor and the generated code was compared with the original so that the code generation coverage could be measured. Such experiments showed that a high percentage of the code could be generated automatically, freeing developers from the most tedious part of encoding.

Keywords: Web Engineering, Frameworks, Model-Driven Development, Code Generation, FrameWeb

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	Application Programming Interface
CDI	Contexts and Dependency Injection
CGI	Common Gateway Interface
CRUD	Create, Retrieve, Update e Delete
DAO	Data Access Object
DERTS	Distributed Embedded Real-Time Systems
DSL	textitDomain Specific Language
DWWS	Desenvolvimento Web e Web Semântica
ERP	Enterprise Resource Planning
FrameWeb	Framework-based Design Method for Web Engineering
IDE	Integrated Development Environment
JPA	Java Persistence API
JSF	Java Server Faces
MDD	Model Driven Development
MVC	Model View Controller
ORM	Object-Relational Mapping
POO	Programação Orientada a Objetos
RIA	Rich Text Applications
SQL	Structured Query Language
UFES	Universidade Federal do Espírito Santo
UFO	Unified Foundational Ontology
UML	Unified Modeling Language
WSI	Web Information Systems
XML	Extensible Markup Language

Sumário

1 Introdução

Vivemos numa sociedade marcada pela era digital, onde os artigos tecnológicos assumem um papel importantíssimo para o funcionamento da sociedade, desde o início da telefonia até a Internet, abrindo assim grandes possibilidades de interconexão e troca de informações entre indivíduos geograficamente distantes de forma rápida e segura, proporcionando uma profunda reorganização do modo como pensamos e agimos na atualidade.

Com o surgimento do computador pessoal nos anos 80 e a popularização da Internet nesse mesmo período, surge uma nova plataforma a ser explorada pelos desenvolvedores: o ambiente *Web*. Inicialmente, os *softwares* propostos para essa plataforma eram simples, sendo desenvolvidos de forma *ad-hoc*. Nessa época surgem, segundo Souza (2007), diversas propostas de métodos para análise, projeto e desenvolvimento de sistemas de informação baseados na *Web* (*Web-based Information Systems* - WISs) (ISAKOWITZ; BIEBER; VITALI, 1998).

Nesta plataforma, a equipe de desenvolvimento pode centralizar o código-fonte no servidor, realizando apenas uma instalação do *software*. Dessa forma, os custos e problemas decorrentes da instalação de programas em computadores de clientes em localidades distantes e/ou arquiteturas distintas, bem como a manutenção e atualização dos mesmos torna-se mais rápida e prática. Aos clientes, havia a possibilidade de acessar as aplicações de qualquer lugar, utilizando dispositivos que estivessem conectados à Internet, necessitando apenas de um navegador para o acesso.

Entretanto, os avanços computacionais possibilitaram o desenvolvimento e utilização de *softwares* mais complexos, gerando uma demanda crescente por sistemas *Web* mais ricos em recursos e, consequentemente, aumentando a complexidade de seu desenvolvimento. Contudo, as técnicas aplicadas ainda eram ineficientes para a elaboração de tais programas. Nesse contexto, inicia-se a adoção de técnicas da Engenharia de Software e sua aplicação no desenvolvimento de sistemas *Web*.

Dentre as técnicas mais difundidas, duas são mais relevantes ao escopo deste trabalho: a utilização de *frameworks* e a linguagem UML.

Os *frameworks* são excelentes ferramentas que visam agilizar o desenvolvimento de *software*. Para Silva (2000), *frameworks* são estruturas de classes que constituem implementações incompletas que, estendidas, permitem produzir diferentes artefatos de software. A grande vantagem desta abordagem é a promoção de reuso de código e projeto, que pode diminuir o tempo e o esforço exigidos na produção de software.

A utilização de *frameworks* visa facilitar o desenvolvimento, uma vez que parte da solução a ser implementada já está desenvolvida e devidamente testada, acelerando o processo (SOUZA, 2007). Existem diversos tipos de *frameworks*, cada qual com sua função específica, mas todos com o mesmo propósito: oferecer um conjunto de soluções padronizadas e testadas, reduzindo a carga de trabalho dos desenvolvedores.

A linguagem UML é uma linguagem de modelagem gráfica utilizada para facilitar e organizar a modelagem de sistemas complexos, em especial, os que utilizam o paradigma de orientação a objetos. Seu uso é muito comum na Engenharia de Software por facilitar a elaboração e compreensão de sistemas complexos (FOWLER, 2014). A UML é adequada para a modelagem de sistemas cuja abrangência poderá incluir sistemas de informação corporativos a serem distribuídos a aplicações baseadas em Web e até sistemas complexos embutidos de tempo real (BOOCH; RUMBAUGH; JACOBSON, 2006).

1.1 Motivação

A cada ano surgem novas demandas para desenvolvimento de sistemas maiores e mais complexos (SOMMERVILLE, 2010). Com isso, cresce a necessidade de se criar ou aperfeiçoar ferramentas e técnicas de reuso de *software*, como os *frameworks* (FRAKES; KANG, 2005), para reduzir o tempo de desenvolvimento, mantendo, porém, o controle e organização esperados na aplicação de técnicas de Engenharia de Software.

Neste contexto, o método *FrameWeb*, proposto por Souza (2007), possui foco num determinado nicho de desenvolvedores que atuam no desenvolvimento de sistemas Web baseados em certos tipos de *frameworks*, a saber: controlador frontal, injeção de dependências e mapeamento objeto/relacional. Os modelos propostos pelo método incorporam os conceitos trazidos por estes *frameworks* para a fase de projeto do sistema, auxiliando a gerenciar a complexidade por trás da implementação da aplicação e a comunicação entre arquitetos e desenvolvedores de software.

FrameWeb é baseado em metodologias e linguagens de modelagem bastante difundidas na área de Engenharia de Software sem, no entanto, impor nenhum processo de desenvolvimento específico. É esperado, apenas, que determinadas atividades comuns à maioria dos processos de software, como levantamento de requisitos, análise, projeto, codificação, testes e implantação, sejam conduzidas pela equipe de desenvolvimento. O método, portanto, sugere um processo de *software* orientado a objetos contendo as fases descritas anteriormente, mas podendo ser estendido e adaptado pela equipe de desenvolvimento.

Por trazerem os conceitos dos *frameworks* explicitamente representados nos modelos, programadores sabem exatamente como implementar os diversos artefatos de código integrados aos respectivos *frameworks*. Muitas destas tarefas, no entanto, podem ser automatizadas com geração de código, aliviando boa parte do esforço de codificação e

permitindo que os desenvolvedores se concentrem mais na lógica do negócio e menos nos processos repetitivos que são encontradas durante o desenvolvimento. Isso nos motivou a construir um gerador de código para o método FrameWeb.

O gerador proposto neste trabalho foi construído com base no metamodelo FW-15, termo que foi definido por Martins (2016), que em seu trabalho apresenta uma evolução do metamodelo originalmente proposto por Souza (2007), e também desenvolve a primeira ferramenta para validação de um modelo construído conforme a definição do metamodelo FW-15. Esse trabalho foi um passo importante para continuidade do *FrameWeb*, sendo base para outros trabalhos relacionados que estão a expandir e/ou melhorar o *framework*.

Um desses trabalhos é o *FrameWeb Editor* (CAMPOS; SOUZA, 2017), uma ferramenta CASE que dá suporte à criação e verificação dos modelos propostos por *FrameWeb* de forma gráfica e baseada no metamodelo FW-15. A ferramenta garante que os modelos construídos são aderentes à linguagem de modelagem definida por *FrameWeb*, o que é fundamental para o bom funcionamento do gerador de código.

Há muitas ferramentas de geração de código disponíveis, inclusive gratuitamente, para aplicações Web baseadas em *frameworks*. No entanto, tais aplicações estão limitadas a um ou alguns *frameworks* específicos. Neste trabalho, portanto, propomos uma ferramenta de geração de código a partir de modelos *FrameWeb* que, sendo baseada em seus metamodelos, pode ser estendida a outros *frameworks* e plataformas, porém, mantendo a linguagem unificada proposta pelo método. Vale ressaltar que até o momento não existe nenhuma ferramenta para transformação de código baseado no metamodelo do *FrameWeb*, de modo a apontar que o gerador irá ajudar na evolução e contribuições no metamodelo durante o seu processo de desenvolvimento.

1.2 Objetivos da Dissertação

O objetivo principal deste trabalho é a produção de um gerador de código focado no uso de modelos gerados pelo *FrameWeb Editor*, batizado de *Gerador FrameWeb*, que possua as seguintes características:

- Incluir em sua construção metodologias de Engenharia de Software Orientada a Objetos e linguagens de programação bem difundidas entre acadêmicos e profissionais da área, de modo a facilitar futuras manutenções;
- Ser construído numa linguagem de programação sem restrições aos sistemas operacionais do mercado e utilizando softwares gratuitos;
- Ser de fácil utilização pelos desenvolvedores, utilizando-se do maior nível de abstração possível para reduzir a curva de aprendizado;

- Incluir suporte a todos os modelos do método *FrameWeb*;
- Ser adaptável a outros editores e/ou, em sua falta, funcionar sem a necessidade do mesmo, por meio de um processo manual do usuário.

Satisfazendo os objetivos acima, o resultado é uma ferramenta que facilita o desenvolvimento de sistemas Web, especialmente na fase inicial de implementação, utilizando técnicas de Engenharia de Software aplicadas ao desenvolvimento Web, *frameworks* e a linguagem de modelagem UML de forma integrada. A partir dos modelos *FrameWeb*, o gerador produz um "esqueleto" do software para ser utilizado e contendo todas as definições propostas no modelo do projeto.

1.3 Método

Para alcançar os objetivos acima, foram realizadas as seguintes atividades:

1. **Levantamento bibliográfico**: foi realizado um levantamento bibliográfico a fim de avaliar artigos científicos e trabalhos acadêmicos que tratassem do desenvolvimento orientado a modelos, utilização de *frameworks*, técnicas de Engenharia de Software aplicadas ao desenvolvimento de *WISs*, linguagem de modelagem e utilização do método *FrameWeb*. Durante a análise, surgiu a ideia da construção de uma ferramenta capaz de transformar os modelos *FrameWeb* representados através da linguagem UML no código-fonte de uma aplicação;

2. **Desenvolvimento do gerador**: desenvolvimento do gerador de código em si, envolvendo ajustes no metamodelo de FrameWeb e codificação dos procedimentos de transformação de modelo para código. Vale ressaltar que, por razões pragmáticas, o gerador de códigos foi produzido em *CSharp*, enquanto o *FrameWeb Editor* foi desenvolvido em Java, como um *plug-in* a ser utilizado na IDE Eclipse;[1]

3. **Definição dos modelos de exemplo**: criação do modelo *FrameWeb* correspondente ao projeto selecionado, utilizando-se dos mesmos *frameworks* e linguagem de programação que foram previamente utilizados;

4. **Escolha do projeto**: como forma de validação, foram selecionados dois projetos, o primeiro desenvolvido pelos alunos de graduação e pós-graduação da UFES, na disciplina de Desenvolvimento Web e Web Semântica (DWWS) e o segundo foi de forma aleatória, obedecendo apenas os critérios de tecnologias suportas pelo método *FrameWeb*. De modo ao final fosse possível comparar resultados e avaliar as vantagens de utilização do método *FrameWeb*;

[1] Eclipse, <https://www.eclipse.org>

5. **Uso do gerador**: após o desenvolvimento do gerador, seu uso foi posto a prova no desenvolvimento do projeto previamente selecionado. O modelo definido no trabalho dos alunos da disciplina de Desenvolvimento Web e Web Semântica (DWWS) foram utilizados como base pelo *FrameWeb Editor*, dando origem ao arquivo .frameweb contendo sua representação. Este, por sua vez, foi utilizado como entrada pelo gerador de código e processado, produzindo como saída o código-fonte da aplicação;

6. **Avaliação**: o código-fonte gerado pelo método descrito no item anterior foi comparado com a aplicação desenvolvida anteriormente e que serve como referencial para este documento. Foram analisadas as semelhanças entre os códigos existentes e os gerados a partir dos modelos. As similaridades encontradas foram tabuladas para melhor compreensão dos resultados.

Vale ressaltar que a base para este projeto foram as seguintes publicações:

- Souza (2007),"FrameWeb: um Método baseado em Frameworks para o Projeto de Sistemas de Informação Web". Dissertação de Mestrado, Universidade Federal do Espírito Santo 2007 — proposta inicial do método *FrameWeb*;

- Martins e Souza (2015), "A Model-Driven Approach for the Design of Web Information Systems based on Frameworks". Proc. of the 21st Brazilian Symposium on Multimedia and the Web, Manaus, AM, Brazil, 2015, pp. 41-48 — proposta do metamodelo FW-15 que define a linguagem de modelagem *FrameWeb*;

- Celino et al. (2016), "A Framework-based Approach for the Integration of Web-based Information Systems on the Semantic Web". Proc. of the 22nd Brazilian Symposium on Multimedia and the Web, Teresina, PI, Brazil, 2016, pp. 231-238 — extensão do método *FrameWeb* para modelagem de vocabulários da Web Semântica e sua ligação com as classes do sistema;

- Campos e Souza (2017), "FrameWeb Editor: Uma Ferramenta CASE para suporte ao Método FrameWeb". Anais do 16º Workshop de Ferramentas e Aplicações, 23º Simpósio Brasileiro de Sistemas Multimedia e Web, Gramado, RS, Brazil, 2017, pp. 199-203 — proposta de um editor gráfico para modelos *FrameWeb* baseado no metamodelo FW-15;

- Almeida, Campos e Souza (2017) "A Model-Driven Approach for Code Generation forWeb-based Information Systems Built with Frameworks". Proc. of the 23rd Brazilian Symposium on Multimedia and the Web, Gramado, RS, Brazil, 2017, pp. 245-252 — publicação do autor desta dissertação, que propõe um gerador de código para modelos *FrameWeb* baseado no metamodelo FW-15.

1.4 Organização da Dissertação

Neste capítulo inicial se apresenta o contexto e a motivação deste trabalho, além dos objetivos e o método para alcançá-los nesta pesquisa. Além disso, esta dissertação é organizada como descrito a seguir:

Capítulo 2 - Revisão da Literatura: descreve a evolução do desenvolvimento de sistemas *Web*, o surgimento dos *frameworks*, o conceito de desenvolvimento orientado a objetos, a utilização da linguagem UML e o funcionamento do método *FrameWeb*;

Capítulo 3 - Proposta do Trabalho: apresenta o gerador de código FrameWeb, explicando seu funcionamento, as interfaces com o usuário desenvolvidas e as modificações no metamodelo FrameWeb necessárias para seu desenvolvimento;

Capítulo 4 - Avaliação do Trabalho: demonstra os resultados obtidos com o uso do gerador de código no desenvolvimento das aplicações propostas;

Capítulo 5 - Considerações Finais: apresenta as conclusões baseadas nos resultados obtidos e as propostas de melhoria para futuras versões.

2 Revisão da Literatura

Neste capítulo são apresentados os resultados da revisão literária feita durante o curso deste trabalho. Na Seção 2.1 é descrito como ocorreu a evolução das técnicas de desenvolvimento de sistemas Web até o surgimento dos *frameworks*. A Seção 2.2 fala sobre o Desenvolvimento Orientado a Modelos, seus conceitos e sua importância de sua aplicação no desenvolvimento de software. A Seção 2.3 fala sobre a linguagem de modelagem UML e sua importância no processo de desenvolvimento de sistemas. A Seção 2.4 descreve o método FrameWeb e seu funcionamento. Na Seção 2.5 é abordado o uso do *Editor FrameWeb* como ferramenta fundamental para criação dos modelos.

2.1 Uso de Frameworks no desenvolvimento Web

Com o surgimento e desenvolvimento da *World Wide Web*, páginas e programas começaram a ser desenvolvidas para esta plataforma, dando início a um novo paradigma de desenvolvimento onde as aplicações encontram-se online e são acessíveis não através de programas instalados em computadores locais, mas através de páginas Web, disponibilizadas por servidores e acessíveis de qualquer computador (SOUZA, 2007). Um exemplo desta mudança de paradigma é o sistema Bling.[1] Trata-se de um sistema *Enterprise Resource Planning* (ERP) desenvolvido com o intuito de auxiliar a gestão de micro e pequenas empresas. Entre os recursos disponibilizados, está a integração com lojas virtuais, tornando possível a emissão de notas fiscais com base nos dados compartilhados entre os sistemas. Todos esses recursos devem estar funcionando ininterruptamente, vinte e quatro horas por dia e deve estar disponível a todos os usuários, além de oferecer atualizações em tempo real de tudo o que acontece nos sistemas integrados a ele, entre outras características que demandam uma alta carga de recursos computacionais para manter-se ativo.

Para que *softwares* mais complexos (como o Bling, citado acima) pudessem funcionar com a mesma eficiência e disponibilidade de suas versões equivalentes para *desktop*, muitas barreiras tecnológicas de hardware e software precisaram ser superadas. De acordo com Pressman (2005), o limite de volume de dados que podem trafegar na rede, disponibilidade de conexão com a Internet, sobrecarga do servidor quando há um grande volume de acessos, lentidão na atualização das informações e a vulnerabilidade dos dados (em caso de acesso não autorizado pela rede) são exemplos dessas limitações, advindas de características inerentes ao ambiente em questão.

Segundo Souza (2007), tais características, juntamente com a complexidade dos

[1] Bling, <https://www.bling.com.br/home>.

sistemas Web conduziram os desenvolvedores a aplicar conceitos da Engenharia de Software no desenvolvimento de sistemas Web, criando assim a *Engenharia Web*, a fim de obter mais rapidez de qualidade nos processos de desenvolvimento, implantação e manutenção dos sistemas desenvolvidos.

A criação do *Common Gateway Interface* (CGI) trouxe mudanças significativas para o desenvolvimento de sistemas Web por possibilitar a utilização de linguagens de programação de forma a tornar as páginas Web mais dinâmicas e, consequentemente, possibilitou seu uso no desenvolvimento de sistemas Web. Outras linguagens de programação foram desenvolvidas ou adaptadas para a utilização no ambiente Web, mas sua disponibilidade não era suficiente para atender às necessidades dos desenvolvedores. Muitos trabalhos eram feitos sem padronização e muitos componentes tinham que ser refeitos a cada projeto, elevando o tempo e os custos de desenvolvimento e tornando os sistemas mais suscetíveis a erros. Entretanto, a crescente complexidade dos sistemas exigia o natural desenvolvimento de técnicas que conferissem maior rapidez e confiabilidade no processo de desenvolvimento das aplicações Web. Surge, nesse contexto o uso de *frameworks* (SOUZA, 2007).

De acordo com Maldonado et al. (2002), nas últimas décadas, diversos *frameworks* foram desenvolvidos. Cada qual com sua especialidade, porém, buscando melhorar a produtividade, qualidade e a manutenibilidade do sistema.

A definição de *framework* é ampla. Para Maldonado et al. (2002), *frameworks* são conjuntos de objetos que colaboram entre si na execução de um conjunto de responsabilidades, reutilizando análise, projeto e código. Já Minetto (2007) descreve os *frameworks* como sendo uma coleção de códigos-fonte, classes, funções, técnicas e metodologias que facilitam o desenvolvimento de softwares. Seu uso permite que sistemas complexos possam ser divididos em pequenos sistemas que se integram em uma estrutura comum. Muitos componentes inerentes ao desenvolvimento do sistema já estão parcial ou totalmente desenvolvidos, já foram testados e estão prontos para funcionar, permitindo que o mesmo seja reutilizado em diversos projetos, diminuindo os riscos de erros e agregando maior padronização aos projetos que o utilizam. *Frameworks* também permitem a abstração de diversos componentes (como por exemplo, os mecanismos de autenticação de usuário), permitindo que a equipe de desenvolvimento se concentre apenas em desenvolver o que é realmente necessário, aplicando as regras de negócio na composição da estrutura pré-definida para a obtenção do sistema final.

Segundo Minetto (2007), o uso de convenções, classes e bibliotecas dos *frameworks* facilita o desenvolvimento e manutenção do código, pois permite que tarefas repetitivas possam ser automatizadas através de ferramentas contidas no próprio *framework*.

2.1.1 Tipos de Frameworks

Existem diversos tipos de *frameworks* disponíveis para serem utilizados no desenvolvimento de sistemas Web, cada qual com suas características e peculiaridades. Para melhor compreensão do objetivo proposto neste documento, apenas os tipos utilizados no mesmo serão descritos.

- **Frameworks MVC:** baseados na arquitetura MVC (*Model View Controller*) (JOHNSON et al., 1995), são *frameworks* desenvolvidos com o intuito de manter a lógica da aplicação, a interface do usuário e o fluxo da aplicação separados, permitindo o acesso da lógica de negócios através de interfaces distintas. Apesar de MVC ser um nome muito difundido, o nome correto para esse padrão arquitetônico é "Controlador Frontal" (*Front Controller*) (PERUCH, 2007).

- **Frameworks RIA:** *Rich Internet Applications* são aplicações Web que contém características e funcionalidades de aplicações *desktop* tradicionais (PERUCH, 2007). Nesse tipo de aplicação, boa parte do processamento é feita não pelo servidor, mas pelo computador do cliente. Segundo Peruch (2007), esse processamento é feito por um *Client Engine* que cada *framework* implementa à sua maneira.

- **Frameworks de Decoração:** proveem uma classe que intercepta as requisições e antes de enviar a resposta aos clientes, eles modificam as páginas originais com o *layout* apropriado a fim de gerar uma página final decorada.

- **Frameworks de Mapeamento Objeto/Relacional:** de acordo com Peruch (2007), a técnica de mapeamento objeto/relacional ou *Object/Relational Mapping* (ORM) consiste em mapear uma representação de dados para um modelo relacional dos dados baseados em *Structured Query Language* (SQL), ou seja, trata-se de uma persistência automática e transparente entre as classes de uma aplicação e as tabelas de banco de dados relacionais.

- **Frameworks de Injeção de Dependências:** trata-se de um *framework* onde a responsabilidade de configurar dependências entre classes é assumida por um terceiro objeto. De acordo com Peruch (2007), nesta solução, as dependências entre os módulos não são definidas pela programação, mas sim pela configuração de uma infra-estrutura de *software* (*container*) no qual as dependências são "injetadas". Sua utilização permite um acoplamento mais fraco entre as classes e, permitindo maior reuso.

A escolha destes *frameworks* não está ligado diretamente a este trabalho, e sim a implementação do metamodelo *FrameWeb*, que por usa vez realiza algumas restrições para atender um nicho de tecnologias que podem ser utilizadas em conjunto com os seus

modelos.

2.2 Desenvolvimento Orientado a Modelo

Segundo Medeiros (2009), muitos sistemas ainda são desenvolvidos de maneira *ad hoc*, sem a aplicação de técnicas de engenharia mais sofisticadas, ainda que o uso destas técnicas aumentem consideravelmente a qualidade e diminuam o tempo necessário para desenvolvimento do *software*. Dentre estas metodologias está o Desenvolvimento Orientado a Modelo (MDD — *Model Driven Development*). Trata-se de uma metodologia que tem como objetivo aumentar o nível de abstração do processo de desenvolvimento de software e automatizar as tarefas manuais. A mudança de perspectiva trazida pela aplicação da metodologia MDD é capaz de melhorar significativamente a qualidade do software produzido, pois reduz sua complexidade e aumenta a possibilidade de reutilização dos componentes, além de aumentar produtividade dos desenvolvedores. Também é possível reduzir o tempo necessário para o desenvolvimento e, ao mesmo tempo, garantir a coerência entre especificação e implementação.

2.2.1 Conceitos da Metodologia Dirigida a Modelos

A abordagem MDD (*Model-Driven Development*) (PASTOR et al., 2008) tem o seu uso na busca de se obter o modelo abstrato que se tornará fundamental para a geração de código e longevidade do projeto. Ela se torna a base principal para manutenção e, na verdade, pode ser a única base para manutenção. Na MDD, os pontos de vista e as visualizações são utilizados para separar preocupações, por exemplo, para lidar com a estrutura lógica independentemente da estrutura física.

Dentro da abordagem MDD, os principais conceitos são os de modelos, metamodelos e hierarquia de metamodelos (MEDEIROS, 2009). Em um processo MDD, os modelos são especificados com uma clara sintaxe abstrata, com regras bem definidas para a sua interpretação. Isso os torna mais facilmente processáveis por máquinas, o que, por sua vez, permite ferramentas para criação de modelos, transformação, validação, etc.

Desta forma o MDD propõe que desenvolvimento, manutenção e evolução sejam realizadas diretamente na etapa de modelagem, algumas características dos modelos:

- são independentes de *software* (da mesma forma que um código de alto nível é independente de *hardware*);
- podem ser compilados para várias linguagens de programação, i.e., gerar código para o sistema;
- podem ser parcialmente ou totalmente reusados em diferentes contextos.

Os metamodelos são modelos de um conjunto de modelos, usados para descrever a semântica de um modelo, podendo portanto, ser definido como um modelo de um conjunto de modelos (MEDEIROS, 2009). De acordo com Souza (2016), os modelos de sistemas Web definem quais elementos devem existir no mesmo, podendo usar uma linguagem gráfica para defini-los. Tal linguagem também pode ser descrita por um modelo de linguagem, que descreve os elementos a serem usados. A este modelo dá-se a definição de metamodelo.

De acordo com Medeiros (2009), podemos imaginar um modelo que descreve a semântica de um metamodelo, como um meta-metamodelo. Os meta-metamodelos tem a tendência de se auto descreverem, evitando a propagação infinita do termo *meta*, assim sendo, temos quatro níveis de (meta) modelos, formando uma hierarquia: os modelos propriamente ditos, os metamodelos, os meta-metamodelos. O quarto nível representa o sistema real.

A abordagem MDD para o desenvolvimento de software permite que os desenvolvedores atuem juntos em um projeto, mesmo que seus níveis de experiência individuais variem muito. Ela permite que uma empresa maximize o trabalho efetivo em um projeto enquanto minimiza a sobrecarga de trabalho necessária para produzir software que pode ser validado pelos usuários finais no menor tempo possível.

2.2.2 Manipulação de modelos e seus desafios

O desenvolvimento de software utilizando a metodologia MDD apresenta desafios a serem superados. Segundo Medeiros (2009), eles estão ligados à criação, análise, utilização e transformações de modelos e à utilização de modelos durante a execução.

Para Medeiros (2009), o primeiro grande desafio está relacionado à propagação das modificações sofridas por um software aos modelos utilizados em sua geração. O segundo desafio é conseguir realizar a integração entre o código manual, o código herdado e o código gerado. O terceiro desafio é a capacidade de supervisionar e gerir diversos aspectos do sistema em tempo de execução utilizando modelos.

Para Corrêa (2009), controlar a evolução e as consequentes modificações realizadas nos modelos é uma tarefa mais complexa do que o controle aplicado ao código fonte em virtude da necessidade de controlar as alterações sofridas por modelos inter-relacionados, levando-se em consideração não apenas as suas versões, mas a consistência entre eles. Depois de gerados, os modelos podem sofrer alterações independentes ao longo do tempo. Essas alterações podem ocorrer em diversos níveis, seja em função da modificação de requisitos do *software*, seja por detalhes relacionados ao código fonte. Essas modificações podem gerar inconsistência em modelos inter-relacionados, caracterizada pela ausência de um elemento comum aos modelos ou pela diferença entre propriedades desses elementos. Para evitar tal inconsistência, a evolução dos modelos deve levar em consideração a inter-relação

entre os mesmos, entretanto, manter a sincronia entre os modelos manualmente é uma tarefa dispendiosa. Não obstante, considerando que haja uma ferramenta capaz de manter o sincronismo e consistência entre os modelos, existe a possibilidade de, por algum motivo, esta não ser utilizada corretamente pelos desenvolvedores. Ambientes de desenvolvimento distribuídos tornam-se um agravante neste cenário, permitindo a utilização de ferramentas diferentes para manipulação dos modelos, aumentando o grau de inconsistência entre os mesmos.

Durante a fase inicial do desenvolvimento de software, alterações de requisitos são eventualmente necessárias. Tais alterações, quando realizadas nos metamodelos, devem ser replicadas a todos os modelos dependentes dele e ao código gerado através destes modelos. Tal tarefa deve ser realizada manualmente e pode acarretar em erros e inconsistências. Assim como a atualização, a depuração também torna-se uma tarefa difícil por não haver um modo eficiente de realizá-la. Durante a pesquisa, foi constatado que a ferramenta Rhapsody[2] é capaz de realizar a depuração a nível de modelo, porém, diferentes linguagens de destino utilizadas pela ferramenta (e o próprio funcionamento da mesma) impedem que este processo seja abstraído, fazendo com que os desenvolvedores precisem conhecer detalhes e limitações sobre o funcionamento da ferramenta de geração, o que eleva o grau de dificuldade e, eventualmente, a curva de aprendizado, além de comprometer a eficiência de sua execução.

O desenvolvimento de sistemas através de linguagens específicas de domínio (DSL — *Domain Specific Language*) é um processo complexo. As ferramentas de meta-edição geram todo o ambiente necessário (editor, verificador, gerador, etc) para a utilização da linguagem DSL, a fim de tornar a tarefa mais simples. Dentre as ferramentas de meta-edição disponíveis, o conjunto de *plug-ins* do projeto *Eclipse Modeling*,[3] o *MetaEdit+*[4] e o *Microsoft DSL Tools*.[5]

2.3 Modelagem UML

A linguagem UML (Unified Modeling Language - Linguagem de Modelagem Unificada) é uma linguagem gráfica que permite a visualização, especificação, construção e documentação de artefatos de sistemas complexos de software (BOOCH; RUMBAUGH; JACOBSON, 2006). Através da UML é possível descrever graficamente os elementos que compõem o sistema a ser desenvolvido, oferecendo aos desenvolvedores uma forma prática e simples de visualizar seus diversos componentes, abrangendo desde banco de dados até os componentes reutilizáveis.

[2] Rhapsody, <https://www.ibm.com/us-en/marketplace/rational-rhapsody>
[3] Eclipse Modeling, <http://www.eclipse.org/modeling/>
[4] MetaEdit, <http://www.metacase.com>
[5] Microsoft DSL Tools, <http://code.msdn.microsoft.com/DSLToolsLab>

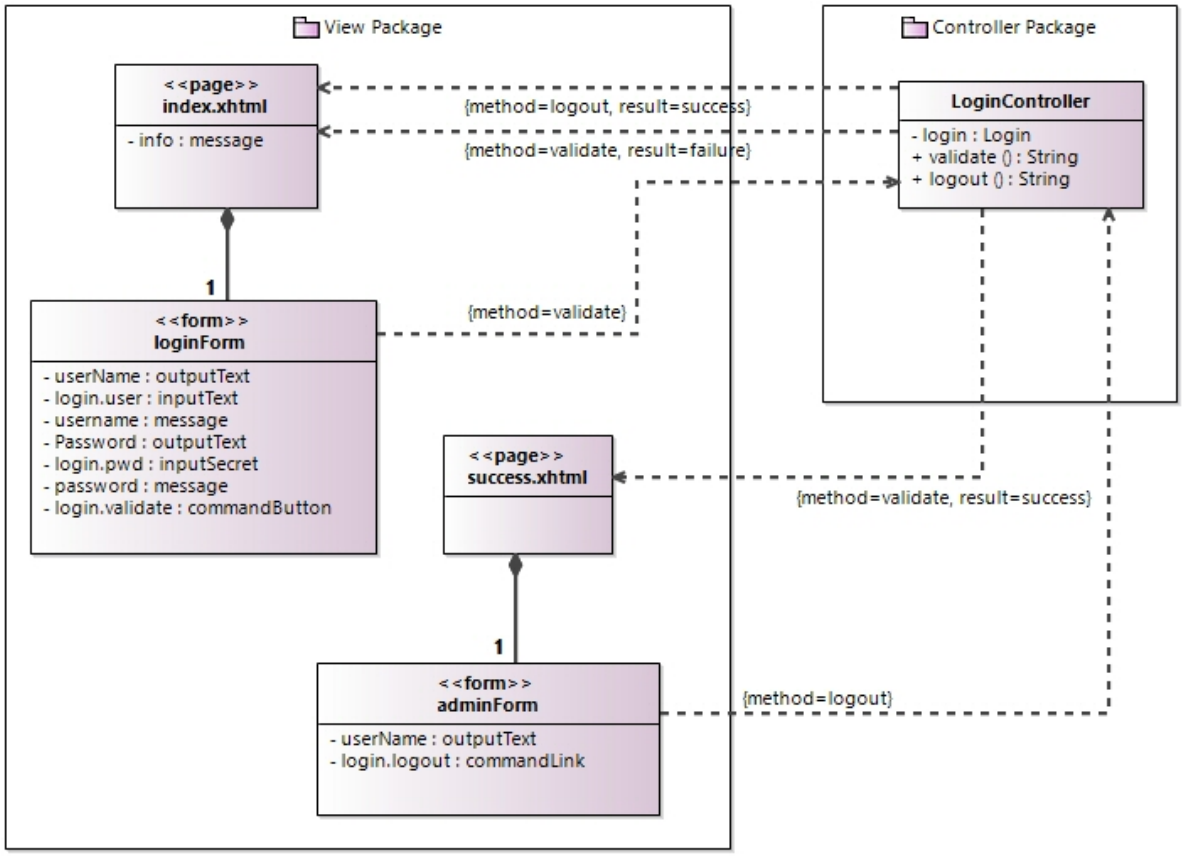

Figura 1 – Modelo de Navegação com JSF

A Figura 1 traz um exemplo de modelagem usando UML seguindo as definições do método *FrameWeb*, onde o modelo de navegação é representado. As seções *View Package* e *Controller Package* indicam quais são as entidades que representam as *views* e o *controller* respectivamente. Dentro de cada entidade encontramos seus atributos. As linhas e setas indicam os relacionamentos entre as entidades. Associadas a elas, estão os métodos responsáveis por esse relacionamento. Como podemos notar, esse diagrama explica em detalhes e, ao mesmo tempo, de forma simples e intuitiva quais são os elementos que compõem o modelo de navegação representado, seus relacionamentos e suas atribuições.

Para a correta representação dos elementos componentes do software, é necessário que a UML possua uma padronização que, por sua vez, é utilizado como referência para o desenvolvimento de ferramentas que permitem sua utilização, como é o caso do Eclipse Modeling.[6]

A especificação UML define uma sintaxe abstrata e uma semântica estática aos diagramas UML por meio de diagramas de (meta) classes e fórmulas OCL.[7] A semântica

[6] Eclipse Modeling, <http://www.eclipse.org/modeling/>

[7] OCL é um acrônimo de Object Constraint Language (ou Linguagem para Especificação de Restrições em Objetos, em português). É uma linguagem declarativa para descrever as regras que se aplicam aos modelos UML desenvolvida na IBM e que agora é parte do padrão UML

dinâmica (operacional) dos diagramas comportamentais é descrita apenas informalmente em linguagem natural. No entanto, ao usar modelos UML para comunicação entre equipes de desenvolvimento, para documentação de projetos ou como um contrato entre desenvolvedores e clientes, é importante que todos os parceiros concordem com uma interpretação comum. Isso requer uma especificação semântica que capta, de maneira precisa, os recursos estruturais e dinâmicos da linguagem.

Com a linguagem UML, podemos construir modelos que nos permitam visualizar e controlar a arquitetura do sistema, compreender melhor o que está sendo elaborado, expor oportunidades de simplificação ou reaproveitamento e gerenciar riscos inerentes ao desenvolvimento de sistemas (BOOCH; RUMBAUGH; JACOBSON, 2006).

A linguagem de modelagem UML é amplamente utilizada na elaboração da documentação de sistemas complexos, especialmente que seguem o paradigma orientado a objetos. Seu uso é bastante difundido entre os desenvolvedores pela facilidade na compreensão de seus modelos gráficos, que representam os elementos que compõem o *software* de forma detalhada e intuitiva, ajudando a equipe a compreender melhor a complexidade do sistema desenvolvido, orientando a implementação e manutenção do mesmo, bem como o treinamento de novos membros da equipe de desenvolvedores.

2.4 O método FrameWeb

FrameWeb (SOUZA, 2007; MARTINS; SOUZA, 2015) é um método para construção de sistemas de informação Web baseados em *frameworks*. FrameWeb propõe uma arquitetura básica que divide o sistema em camadas (Apresentação, Lógica de Negócio e Acesso a Dados) para uma melhor integração com *frameworks* tipo controlador frontal (ex.: Java Server Faces[8]), injeção de dependências (ex.: Contexts and Dependency Injection for Java[9]) e mapeamento objeto/relacional (ex.: Java Persistence API[10]).

O método propõe ainda uma linguagem de modelagem baseada na UML para a construção de quatro modelos de projeto diferentes (todos baseados no Diagrama de Classe da UML) que trazem os conceitos usados por esses *frameworks* para o estágio de projeto arquitetônico do processo de software:

- **Modelo de Persistência:** representa os objetos de acesso a dados (do padrão DAO (ALUR; CRUPI; MALKS, 2003)) responsáveis pela persistência de objetos de domínio (camada de Acesso a Dados);
- **Modelo de Entidades:** representa classes de domínio e seus metadados para

[8] JSF, <http://jcp.org/en/jsr/detail?id=344>
[9] CDI, <http://jcp.org/en/jsr/detail?id=346>
[10] JPA, <http://jcp.org/en/jsr/detail?id=338>

mapeamento objeto/relacional (camada de Lógica de Negócio);

- **Modelo de Aplicação:** representa as classes que são responsáveis pela implementação das funcionalidades do sistema (camada Lógica de Negócios) e sua relação com as classes de outras camadas (isto é, a injeção de dependências);
- **Modelo de Navegação:** representa os componentes que formam a camada de apresentação, como páginas Web, formulários, etc. (camada de Apresentação) e sua integração com o controlador frontal.

2.4.1 O Metamodelo de FrameWeb

O FrameWeb utiliza a UML como uma linguagem específica de domínio (DSL) para o método, tendo como base o metamodelo UML e adicionando tipos específicos de domínio para construir o metamodelo FrameWeb. Ele é definido em um meta-nível mais alto (M2) e possui dependência do metamodelo UML, conforme a Figura 2, utilizando apenas uma pequena parte do metamodelo UML. A sintaxe independente da estrutura conduz a criação do modelo sob um ponto de vista geral, ou seja, independentemente do conjunto específico de *frameworks* utilizado no projeto. Os modelos criados no nível M1 usando a linguagem FrameWeb precisam seguir as mesmas regras independentes da plataforma.

Cada modelo *FrameWeb* possui uma parte do metamodelo que define sua sintaxe, também mostrada na Figura 2: o modelo de entidades é uma instância da parte do metamodelo de entidades, o modelo de persistência é uma instância do metamodelo de persistência, e assim por diante. Todas as quatro partes dependem da parte do metamodelo correspondente.

Na Figura 3 há um fragmento do metamodelo de navegação, definindo a sintaxe dos modelos de navegação. Suas meta-classes representam componentes da camada de apresentação, por exemplo:

- `Page`: representa uma página da Web dinâmica;
- `UIComponent`: representa o componente formulário de uma página dinâmica;
- `FrontControllerClass`: representa a classe controladora de uma página dinâmica (um elemento do *framework* controlador frontal).

Na Figura 1, há uma instância deste metamodelo, utilizando o JSF como controlador frontal.[11] Os estereótipos do perfil original do *FrameWeb* são usados para identificar a meta-classe de cada classe no modelo, por exemplo:

[11] Este exemplo foi extraído com base no tutorial encontrado no site <http://www.thejavageek.com/

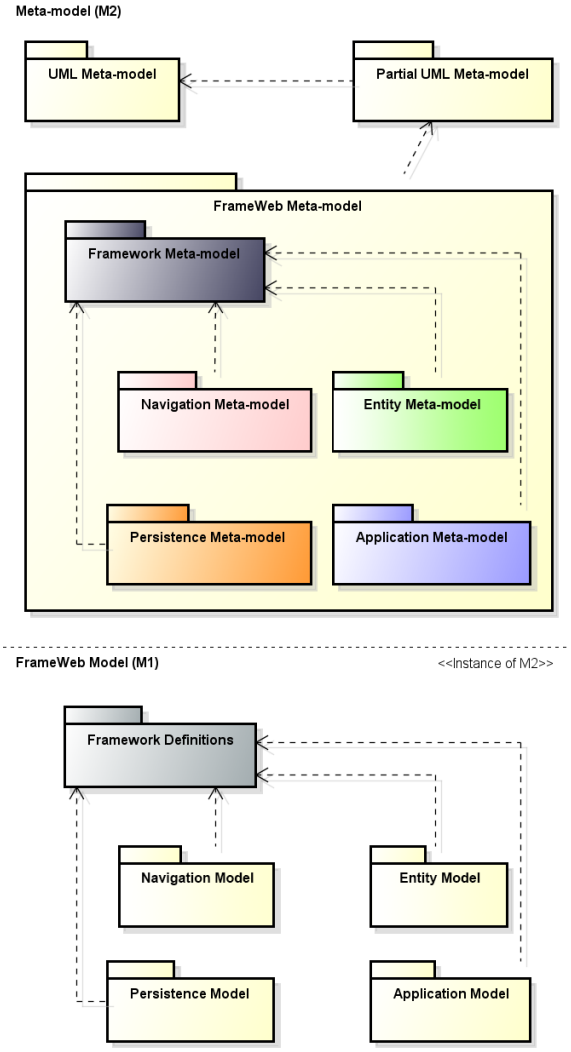

Figura 2 – Níveis de abstração M1 e M2 do FrameWeb (MARTINS; SOUZA, 2015).

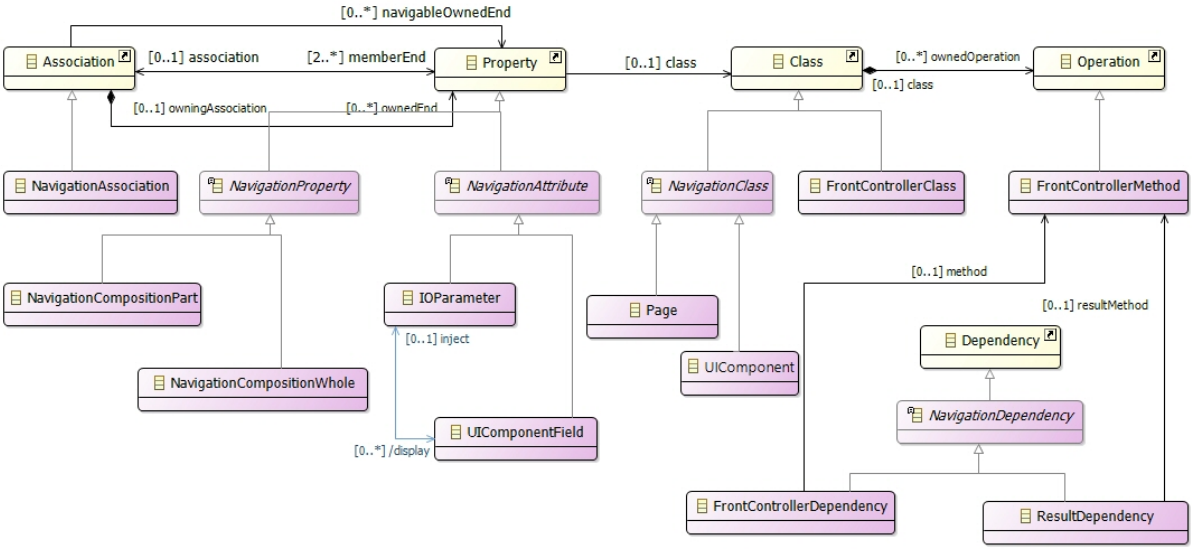

Figura 3 – Um fragmento do Metamodelo de Navegação do *FrameWeb* (MARTINS, 2016).

- `Page`: estereótipo «page»;
- `UIComponent`: estereótipo «form»;
- `FrontControllerClass`: sem estereótipo.

Ainda na Figura 1, há duas páginas (`index.xhtml` e `success.xhtml`), dois formulários (`loginForm` e `adminForm`) e uma classe de controlador (`LoginController`). O metamodelo (M2) especifica como esses componentes (em M1) podem se relacionar entre si, permitindo apenas algumas relações específicas de dependências e composições entre eles.

Como resultado, temos a possibilidade de gerar código automaticamente a partir de modelos, reduzindo o esforço inicial necessário por parte dos desenvolvedores na escrita de código no início do projeto seja ele relacionado a infra-estrutura dos *frameworks* ou módulos próprios.

2.5 Editor FrameWeb

A documentação do *software* é de grande importância para seu desenvolvimento. Ela representa o sistema a ser implementado e o máximo possível de sua complexidade, permite que novos desenvolvedores aprendam sobre seu funcionamento com maior rapidez e facilidade e orienta o desenvolvimento, implementação e manutenção de cada um de seus recursos.

O uso do editor é fundamental para auxiliar na criação dos modelos, pois diferentemente dos editores que trabalham apenas com UML, o *Editor FrameWeb* realiza análises e aplica validações ao modelos de acordo com as definições e restrições do *framework*.

Campos e Souza (2017) propõem o *Editor FrameWeb*, uma ferramenta para criação/edição de modelos baseados nas definições do método *FrameWeb*. O editor utiliza uma linguagem de modelagem com sintaxe definida formalmente por meio dos metamodelos propostos na Seção 2.4.1, servindo como base para a elaboração de uma ferramenta gráfica para criação de modelos válidos nesta linguagem e que serve como base para outras funcionalidades, como a geração de código proposta neste trabalho.

O editor oferece suporte para criação dos quatro tipos de modelos propostos pelo método FrameWeb: *Entity, Persistence, Application* e *Navigation*, além de um nível de projeto que agrega os quatro modelos, pacotes de linguagem e de definições de *frameworks*, como vemos na Figura 4.

A interface principal da ferramenta possui um painel à direita onde são exibidas

2013/12/18/login-application-jsf/>.

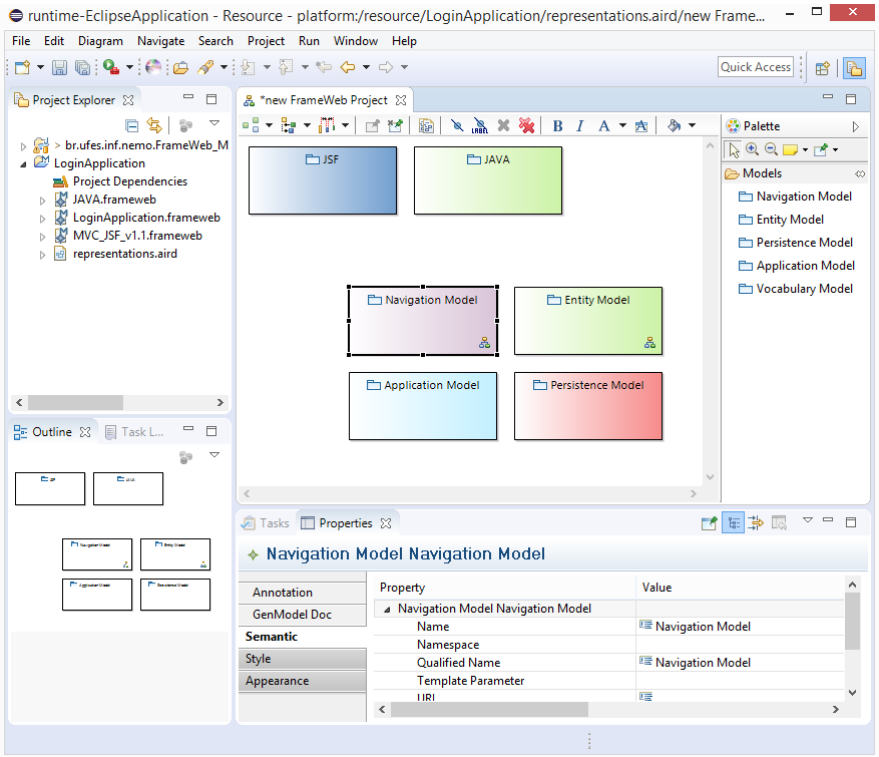

Figura 4 – Visão geral de um WIS modelado com FrameWeb usando o editor.

as opções de componentes que podem ser criados no tipo de modelo em questão. Na parte inferior, há uma lista das propriedades dos componentes que podem ser alteradas. À esquerda, há o *Project Explorer*, para visualização do projeto. O projeto utilizado apresentado na Figura 4 utiliza a plataforma Java e a tecnologia Java Server Faces (JSF) como *framework* Web.

Os arquivos criados pela ferramenta são arquivos XML, salvos no formato `.frameweb`, e são processados pelo gerador de código proposto na Seção 3.1 para a geração do código-fonte. Na Listagem 2.1, podemos ver um fragmento do código `.frameweb` gerado pelo editor, a ser utilizado pelo gerador de código.

Listagem 2.1 – Trecho de arquivo .frameweb gerado pelo FrameWeb Editor

```
<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 Package/loginForm" association="//@compose.0/
                View Package/@packagedElement.5"/>
        </packagedElement>
        <packagedElement xsi:type="frameweb:Page" name="success.xhtml">
            <ownedAttribute xsi:type="frameweb:NavigationCompositionWhole" type="
                //@compose.0/View Package/adminForm"association="//@compose.0/View
```

```
        Package/@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>
```

Esse é um passo fundamental para este trabalho, sendo que o arquivo resultante do editor é utilizado no início do processo de transformado das informações contidas nele em artefatos de código para os desenvolvedores poderem continuar o processo de desenvolvimento. No próximo capítulo, apresentamos o gerador de código desenvolvido neste trabalho.

3 Proposta do Trabalho

Este capítulo apresenta a proposta principal deste trabalho, trazendo uma breve explicação sobre como o método FrameWeb pode ser padronizado para então ser utilizado na elaboração dos softwares descritos nas seções 2.5 e 3.1 que, respectivamente, descrevem o editor visual de modelos e metamodelos FrameWeb — onde é possível realizar a modelagem do sistema utilizando a linguagem UML e os conceitos do FrameWeb — e o gerador de código — que utiliza o arquivo gerado pelo editor para gerar o código-fonte da aplicação. A Seção 3.2 apresenta uma interface Web para o gerador de código, desenvolvida para facilitar a adoção do mesmo. A Seção 3.3 descreve a arquitetura interna do gerador de códigos e a Seção 3.4 apresenta algumas limitações do metamodelo do FrameWeb que, por conseguinte, implicam em limitações também do gerador de código.

A utilização do método FrameWeb no desenvolvimento de sistemas visa tornar mais fácil o gerenciamento da complexidade por trás de sua implementação e manutenção. Entretanto, a relação entre os modelos e metamodelos do método com a implementação final da aplicação permanecem desconexas. A documentação elaborada durante o desenvolvimento inicial do *software* não está diretamente associada à fase de implementação do mesmo, gerando discrepâncias e inconsistências que dificultam a implementação e manutenção do software.

Durante a análise do método FrameWeb, percebeu-se que alguns conceitos poderiam ser aproveitados para o desenvolvimento de um editor e gerador de código, capaz de automatizar a geração do código-fonte da aplicação. Entre esses conceitos está o uso da metodologia de Desenvolvimento Orientado a Modelos (MDD). Essa metodologia é capaz de combinar a linguagem de modelagem UML com a linguagem geradora XML, utilizadas pelo *software* proposto neste documento (MEDEIROS, 2009).

O conceito de transformação de modelos, contido no método MDD, permite que os elementos representados em linguagem de modelagem UML possam ser transformados em modelos de entrada e estes transformados em modelos de saída, obedecendo a hierarquia estabelecida entre eles e os níveis de abstração representados.

A proposta apresentada nesse trabalho é a de desenvolver um *software* gerador de código integrado ao método FrameWeb, capaz de utilizar as mesmas técnicas do Desenvolvimento Orientado a Modelos proposta pelo método para a geração de código do sistema projetado, sendo também extensível a outras plataformas de programação e *frameworks*.

O objetivo é utilizar a estrutura das entidades da linguagem UML utilizadas para representar os elementos componentes do sistema e sua relação com os modelos e

metamodelos do método FrameWeb como orientação no processo de elaboração do código-fonte do sistema. Um trabalho semelhante já fora realizado por Conallen (2003), que demonstra como transformar modelos criados em UML em códigos escritos em linguagem orientada a objetos. O principal diferencial deste trabalho é o uso do metamodelo *FrameWeb*, seguindo todas as suas regras de restrições, que incorporam aos modelos de projeto a arquitetura definida pelo uso dos *frameworks* e suas características.

De forma semelhante ao trabalho de Conallen (2003), o *software* utilizará a estrutura de modelos e metamodelos que compõem o método FrameWeb para orientar-se na construção do código da aplicação. Para isso, tais elementos serão tabulados e padronizados, a fim de permitir a sua correta interpretação pelo *software* durante a elaboração do código.

Durante a geração do código-fonte, o *software* contará com a ajuda de arquivos de configuração, que devem orientar o mesmo na elaboração do código final. Dessa forma, é possível fazer com que o sistema consiga gerar códigos em diferentes linguagens de programação apenas alterando o arquivo de configuração correspondente da maneira correta, orientando corretamente o processo.

O *software* deve promover a integração direta entre os modelos e o código implementado, permitindo uma sincroniza entre o que foi documentado nos diagramas de modelos e representado por meio dos metamodelos e o que foi efetivamente implementado pela equipe de desenvolvimento. A padronização do método FrameWeb e a utilização dos arquivos de configuração, como descrito no parágrafo anterior, deverão promover essa integração de forma automatizada, tornando-a parte do código gerado pelo *software*.

Inicialmente, o *software* proposto neste trabalho será desenvolvido com arquivos de definição de linguagem e de definição de *frameworks* voltados para a utilização em aplicações Web baseadas na linguagem Java e na utilização de seus *frameworks*, havendo a possibilidade de desenvolver o mesmo *software* e seus arquivos de configuração correspondentes para outras linguagens de programação, tarefa que não será detalhada nesse documento, devendo ser objeto de pesquisa em trabalhos futuros.

3.1 Gerador de Código

Trata-se de uma aplicação capaz de interpretar os dados do arquivo gerado pelo *Editor FrameWeb* e transformar em código-fonte necessário para a aplicação. A estrutura do arquivo gerado pelo editor descreve quais são os modelos do sistema e, com esses dados, o gerador identifica quais são as classes, seus atributos e métodos a serem implementados, bem como o relacionamento entre elas. Além disso, também realiza a transformação da parte de visão de um sistema Web, sendo essa uma característica *FrameWeb*, pois, na prática, geralmente não se modela a camada de visão diretamente em UML, essa é uma diferença notável ao se modelar um software seguindo o metamodelo *FrameWeb*.

Uma vez identificados os modelos e seus relacionamentos, inicia-se o processo de transformação dos dados recebidos em código, gerando de fato os arquivos para a aplicação modelada no editor.

Esse processo se inicia quando o gerador recebe o arquivo `.frameweb` como entrada. É realizada, primeiramente, uma conversão em objetos para facilitar o seu uso e, ao final do processo, terá gerado o código-fonte do sistema descrito no mesmo. Construído em CSharp por razões pragmáticas, futuramente deverá ser escrito em Java para permitir maior integração com o editor *FrameWeb*. Entretanto, não há impedimento para a linguagem destino, uma vez que, de acordo com o *FrameWeb*, o mesmo modelo pode representar sistemas em diferentes linguagens e *frameworks*, dependendo das definições de plataforma e *framework* que são carregadas no editor.

Desta forma, construindo um gerador flexível para possibilitar a adição de novas linguagens e *frameworks*, foram elaborados alguns mecanismos de configuração para essa extensão do projeto. O primeiro foi a utilização de um arquivo de configuração, no qual encontramos algumas chaves para configurar caminhos (diretórios) onde se encontram os arquivos e informações necessárias para o funcionamento do gerador. Desta forma foi possível organizar melhor os arquivos e portar o gerador para outras linguagens. Essas chaves se encontram no arquivo `App.Config`,[1] demonstrado na Listagem 3.1.

Listagem 3.1 – Parte do arquivo App.Config, que configura o gerador de código

```
<appSettings>
    <add key="dir_template" value="template\ " />
    <add key="project" value="java_jsf" />
    <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>
```

O significado de cada chave de configuração é explicado na Tabela 1.

A ferramenta *FrameWeb Editor* utiliza arquivos `.frameweb` de definição de *framework* e arquivos de definição de plataforma/linguagem de programação para incorporar aos modelos classes específicas de *frameworks* e classes relacionadas a determinada linguagem/plataforma, respectivamente. Neste estudo de caso, foi adicionado o arquivo `MVC_JSF.frameweb` ao projeto, referente ao *framework* JavaServer Faces (JSF), que contém definições das *tags* utilizadas por este *framework*, permitindo que elas sejam utilizadas nos Modelos de Navegação construídos pela ferramenta. Parte deste arquivo é apresentado na Listagem 3.2.

[1] Este arquivo é padrão para projetos console na plataforma .NET. Nele, é possível definir chaves e configurações que serão acessadas pelo aplicativo sem necessidade de recompilar ou de criar novos arquivos.

Tabela 1 – Informações sobre todas as chaves de configuração.

Chave	Descrição
dir_template	Diretório raiz, onde encontram-se os modelos de projeto e linguagem.
project	Projeto base. Será realizada uma cópia fiel deste projeto para a pasta de saída do código.
lang	Diretório onde se encontram as características da linguagem escolhida. Possui arquivos conforme as entidades existentes no metamodelo.
dir_output_web	Diretório dentro do projeto base onde serão criados os arquivos da camada de visualização.
dir_output_class	Diretório dentro do projeto base onde serão criados os arquivos de acordo com a linguagem escolhida.
ext_class	Extensão dos arquivos de classe. Deve ser utilizado o padrão, conforme a linguagem escolhida (e.g., `.php`, `.cs`, `.java`, etc.)

Listagem 3.2 – Parte do arquivo MVC_JSF.frameweb que representa o perfil do *framework* JSF

```
<packagedElement xsi:type="frameweb:TabLIb" 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"/>
```

Como visto na Listagem 3.2, existe uma propriedade para cada *tag* chamada `codeGenerationTemplate`. Ela informa o caminho do modelo do documento (*template*) que representa o componente, como, por exemplo, para a tag `inputText`:

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

Com base na *tag* acima, temos chaves que serão substituídas pelo gerador durante o processo de transformação. As chaves `FW_ID` e `FW_VALUE` serão substituídas por valores capturados no arquivo `.frameweb` do modelo. Utilizando como exemplo a propriedade `login.user` com o tipo `inputText`, seu resultado seria:

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

Neste exemplo é possível observar que os valores obtidos foram inseridos no lugar das variáveis `FW_ID` e `FW_VALUE`, após serem adequados ao formato de cada atributo.

Ao inciar, o gerador carrega os arquivos de metamodelo da linguagem e *frameworks* que se encontram no diretório previamente configurado pelo usuário, conforme a configuração da chave `dir_profiles` no arquivo `App.Config`. Dessa forma o gerador pode ser utilizado para outras linguagens ou *frameworks*, necessitando que o usuário crie um novo arquivo de perfil para o *framework* ou linguagem desejada e seus respectivos arquivos de modelo (*template*), correspondentes aos elementos criados no perfil. Esse tipo de atualiza-

ção/adição de novas linguagens ou *frameworks*, a priori, é realizada sem a necessidade de alteração no código fonte do editor ou gerador, facilitando assim a possibilidade de sua extensão. Havendo então a necessidade de manutenção no código do gerador, o usuário deverá ter conhecimentos na linguagem de programação C# e entendimento do metamodelo *FrameWeb*.

Tabela 2 – Principais variáveis e respectivas meta-classes

Meta-Classe: **FrontControllerClass**	
Chave	Descrição
FW_CLASS_NAME	Nome da classe criada no modelo.
FW_BEAN_NAME	Nome do *bean* criado no modelo. Um *bean* representa uma entidade, unidade ou modelo do sistema. Ele encapsula as informações necessárias a serem transportadas ao longo das camadas ou módulos.
FW_FRONT_CONTROLLER-_PARAMETERS	Preenche com todos os parâmetros de um determinado controlador.
FW_FRONT_CONTRO-LLER_METHODS	Preenche com todos os métodos de um determinado controlador.
Meta-Classe: **FrontControllerMethod**	
Chave	Descrição
FW_METHOD_RETURN-_TYPE	Tipo especificado para um retorno de método.
FW_METHOD_NAME	Nome do método
Meta-Classe: **IOParameter**	
Chave	Descrição
FW_PARAMETER_TYPE	Tipo especifiado para uma propriedade.
FW_PARAMETER	Nome especificado para uma propriedade
FW_PARAMETER_FIRST-_UPPER	Capitula o tipo especificado para uma propriedade
Meta-classe: **UIComponent**	
Chave	Descrição
FW_BODY	Todo conteúdo gerado durante a criação da página existente no modelo.
FW_ID	Nome do componente criado no modelo.
FW_VALUE	Nome do controlador, mais o nome do componente dentro do formulário, separado por ponto.

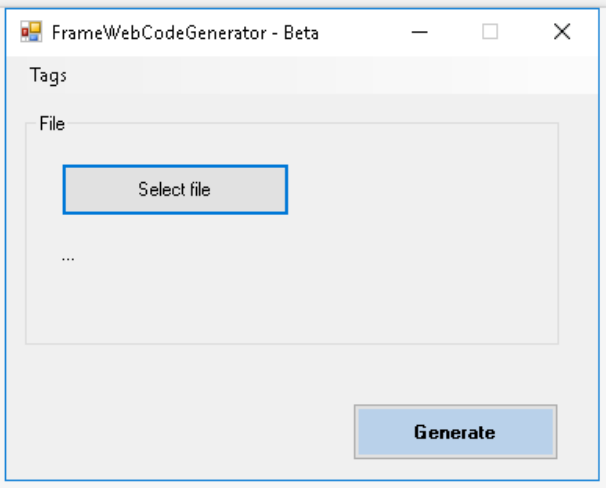

Figura 5 – Aplicativo para facilitar o uso do gerador de código.

A Tabela 2 lista as principais variáveis utilizadas pelo gerador e suas meta-classes correspondentes. Ao encontrar uma instância destas meta-classes em modelos FrameWeb (i.e., arquivos `.frameweb`), o gerador processa o *template* adequado e substitui as variáveis listadas na tabela pelo valor correspondente utilizado no modelo.

O gerador de código oferece suporte aos modelos de Entidades, Navegação, Persistência e Aplicação, realizando a criação dos arquivos de cada classe modelados pelo editor. Seu ciclo de processo é feito da mesma maneira que o feito para o Modelo de Navegação, descrito nesta seção. Há *templates* que refletem o esqueleto de uma classe conforme a linguagem de programação desejada.

Para tornar mais fácil o uso do gerador, há um aplicativo *desktop* (Figura 5), com uma interface simples, no qual o usuário encontra um botão pesquisar e selecionar o arquivo construído com o editor e, após confirmação, tem como resultado o código gerado.

Esse aplicativo possui uma outra funcionalidade: editar os arquivos de *template* de cada elemento descrito no arquivo de perfil. Ao clicar em **tags** > **edit**, será aberta uma tela conforme a Figura 6. Nela, é possível observar que há a possibilidade de selecionar o arquivo desejado e realizar um filtro por *tags*. Ao efetuar dois cliques sobre a linha do elemento, uma nova tela surge. Nela, é possível realizar a edição do conteúdo do *template* pré-definido, possibilitando assim sua customização conforme necessidade do desenvolvedor. A Figura 7 demostra tal ação.

A vantagem de alterar o *template* padrão é poder caracterizar ou facilitar a elaboração de customização de telas, por exemplo, definindo uma classe *CSS* para determinados elementos (botões, por exemplo) gerados pelo editor.

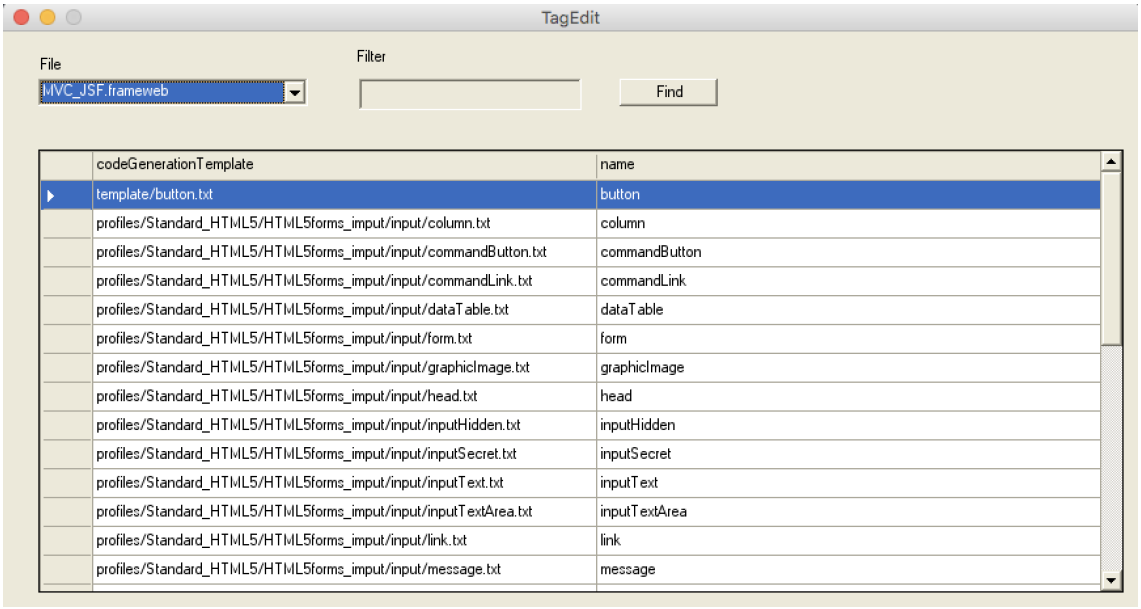

Figura 6 – Tela onde são exibidos todos os elementos de um determinado arquivo de perfil *FrameWeb*.

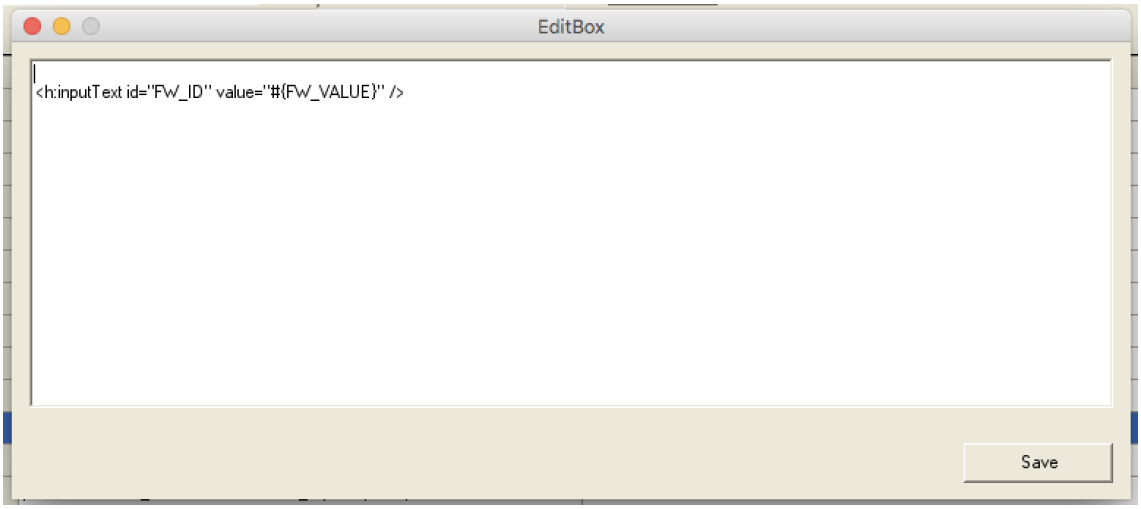

Figura 7 – Nessa tela é possível alterar um arquivo de *template* relacionado a um elemento do perfil *FrameWeb*.

3.2 Gerador de Código Web

Além da versão desktop, foi construída uma versão Web do gerador para transformação do modelo em código. O *Gerador de Código Web* facilita a utilização do componente de transformação por meio de uma interface Web. Sem a necessidade de configuração ou instalação de quaisquer pacotes ou softwares, o serviço encontra-se disponível em <https://frameweb.herokuapp.com>. Como mostra a Figura 8, ao acessar o site é exibido um campo para o usuário realizar a seleção e envio de seu arquivo modelo, construído no *Editor FrameWeb*.

Após realizar o envio do arquivo, pressionando o botão *Execute* o sistema irá

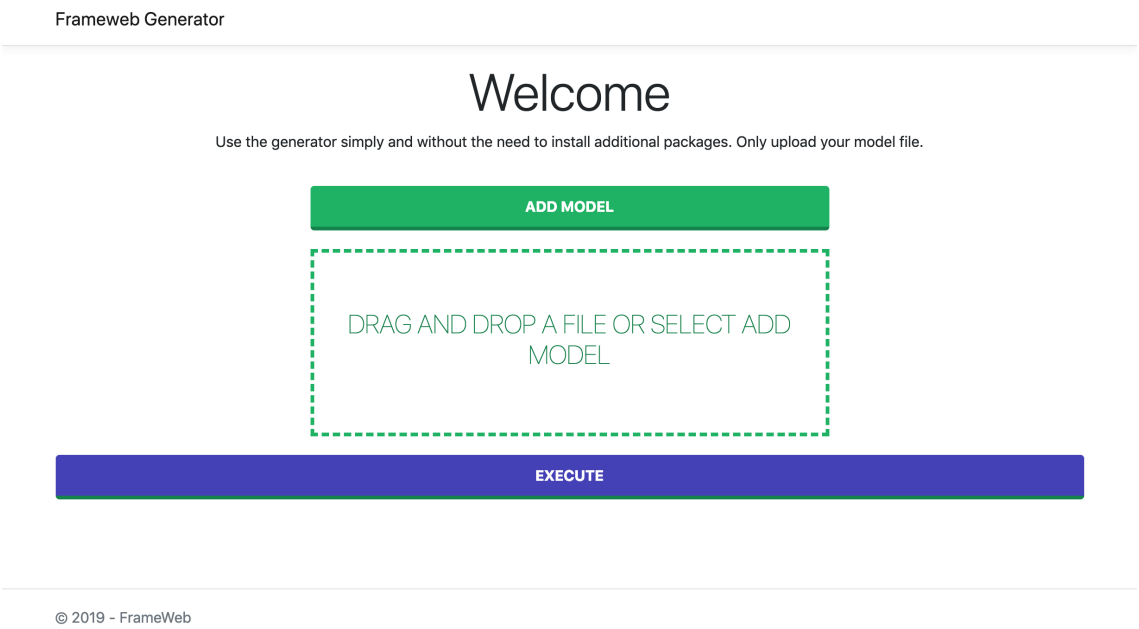

Figura 8 – Imagem do site *Gerador de Código Web*.

processar o modelo enviado e seu retorno é um arquivo compactado contendo o código fonte da aplicação.

3.3 Documentação do Gerador

Conforme evolução do código do gerador, houve modificações até chegar a sua versão atual, disponível no GitHub.[2] Em sua atual versão, após refatorações e melhorias de código, uma base enxuta e consolidada da aplicação foi obtida. Nessa seção será descrita a atual visão sobre a modelagem do software deste trabalho.

Na Figura 9, temos o modelo que define a base do gerador, onde é possível observar que cada modelo do método *FrameWeb* (*Model, Navigation, Application e Persistence*) é implementado ao estender a classe abstrata `Processor`, que por sua vez implementa a interface `IProcessor` de modo que, numa necessidade de incluir um novo modelo do método *FrameWeb* ou uma nova característica não definida no metamodelo, seria necessário estender a classe `Processor` e elaborar a nova parte de código na nova classe com os passos a serem executados durante o processo de transformação/geração de código.

A classe `Config` possui informações referentes aos parâmetros de configuração da aplicação e são acessíveis pelas classes que estendem a classe abstrata `Processor`. Nela há informações como diretório de saída dos arquivos gerados, linguagem de programação utilizada pelo usuário, entre outros. Essas informações são obtidas ao carregar o arquivo de configuração da aplicação, descrito na Tabela 1.

[2] <https://github.com/nilber/FrameWebCodeGenerator>

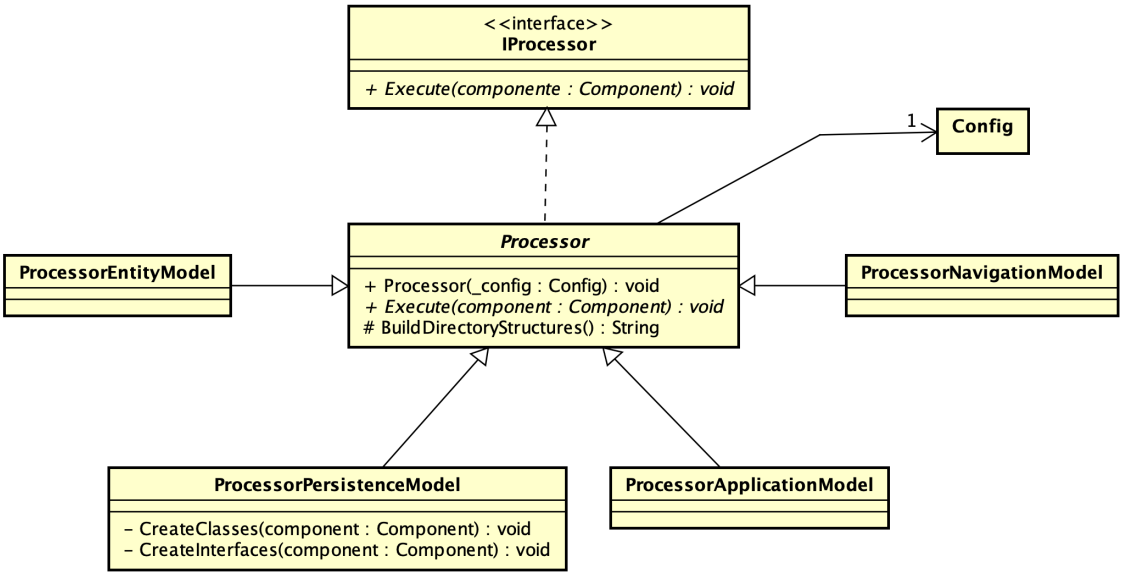

Figura 9 – Diagrama de classe do *Gerador FrameWeb*.

Aprofundando no modelo do gerador, temos a Figura 10, na qual temos as classes: `Profile`, `Config` e `Component`, sendo:

- `Profile`: classe utilizada com espelho de um arquivo de perfil *FrameWeb*. Nele podemos encontrar, por exemplo, a propriedade `codeGenerationTemplate`, que contém o caminho para o arquivo de *template* de um determinado elemento ou tag;

- `Config`: classe utilizada para armazenar informações sobre a configuração da aplicação, como as descritas na Listagem 3.1;

- `Component`: classe utilizada com espelho de um arquivo `.frameweb` de um projeto criado no editor. Um dos primeiros passos do gerador é criar uma instância dessa classe e preenchê-la com as informações oriundas do arquivo. Após esse procedimento, ao invés de utilizar a navegação por um arquivo XML, são utilizados os conceitos de orientação a objetos e lista encadeada para navegar entre as propriedades e elementos filhos do arquivo XML. Com isso, obteve-se uma maior facilidade em extrair e processar dados contidos no arquivo XML.

Os auto relacionamentos existentes em `Component` e `Profile` se devem ao objeto representar arquivo de entrada (`.frameweb`) que se encontra na linguagem XML, ou seja, existem nós (tags) internos que por sua vez possuem propriedades relacionadas ao nó (tag). Formando assim uma lista encadeada de objetos, que podem ser acessados quando necessário pela aplicação para obtenção e/ou navegação da estrutura XML, porém diretamente por um objeto definido, facilitando assim o uso durante a construção e futuras manutenções do projeto.

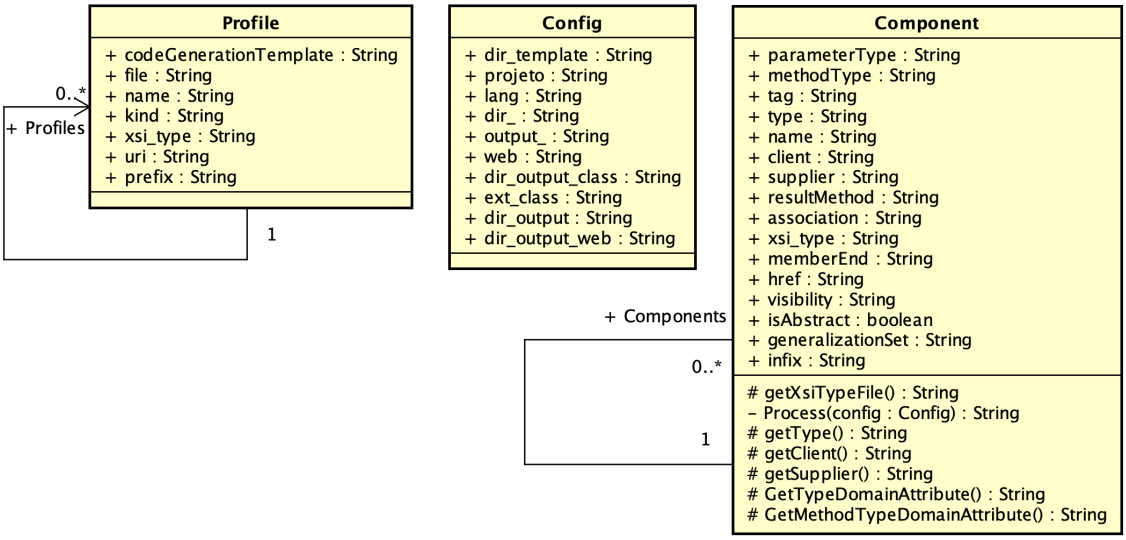

Figura 10 – Diagrama de classe do *Gerador FrameWeb*.

Abaixo a descrição do fluxo para exemplificar de forma mais detalhada a operação exercida pelo gerador:

1. Inicia o sistema;
2. Realiza a instanciação o objeto **Config** e o preenche após leitura do arquivo de configuração `App.Config`;
3. Remove diretório destino do projeto caso exista;
4. Cria uma nova estrutura de diretórios conforme configuração;
5. Copia arquivos padrão (definidos no arquivo de configuração) para o diretório destino do projeto;
6. Realiza a leitura dos arquivos de perfil do FrameWeb;
7. Realiza a leitura do arquivo `.frameweb` contendo os modelos da aplicação a ser construída. A cada modelo (Persistência, Navegação, etc.) lido, ao final da leitura da parte correspondente ao modelo no arquivo é realizada a chamada do método `Execute` referente àquela instância do modelo;
8. Neste momento todas as operações inerentes ao modelo são realizadas utilizando leitura de objetos `Component` e `Profile`, ao final são gerados os arquivos conforme *template* definido para cada tipo de componente;
9. Sistema finaliza operação e abre o diretório de destino.

3.4 Modificações no metamodelo FrameWeb

Durante a etapa de criação do editor e gerador de código, surgiu a necessidade da realização de alterações no metamodelo do FrameWeb para melhor adequação, entre elas:

- Adição de um atributo chamado `version` na classe `FrameworkProfile`, para representar a versão do *framework*;
- Remoção da classe `Result` e criação de uma nova propriedade na classe `ResultConstraint` para representá-la, pois o valor atribuído ao resultado referente a uma `ResultDependency` sempre será uma *string*;
- Alteração da classe `DomainAttribute` de abstrata para concreta, pois da forma como havia sido implementada, só era possível a criação de atributos específicos, como `LOBattribute, VersionAttribute` e demais tipos definidos no metamodelo;
- Adição do atributo `CodeGenerationTemplate` à classe `Tag`;
- Adição dos atributos `collection`, `fetch`, `order` e `cascade` à classe `DomainConstraints`;
- Adição do atributo `methodType` às classes `ServiceMethod`, `FrontControllerMethod`, `DAOMethod` e `DomainMethod` para a representação do tipo de retorno.

Essas mudanças foram realizadas para obtenção de um melhor resultado e fidelidade das informações registradas no modelo provenientes da aplicação a ser construídas, ou seja, uma revisão primeira versão `FW-15` onde o foco era formalização do método `FrameWeb` numa estrutura de metamodelo. Vale ressaltar que o *Editor FrameWeb* também foi responsável por algumas das alterações no metamodelo listadas acima, para que pudesse repassá-las ao modelo gerado e posteriormente serem processadas pelo *Gerador FrameWeb*.

No geral as mudanças foram necessárias para que mais informações fossem capturadas pelo *Editor FrameWeb* de modo a ficarem acessíveis ao *Gerador FrameWeb*, sendo esta uma das formas utilizadas para melhorar e ampliar a cobertura do projeto proposto. Como exemplo, a "Adição do atributo `CodeGenerationTemplate` à classe `Tag`" permitiu tornar dinâmico o acesso ao *template* de cada componente (tag) existente numa determinada linguagem. Este tipo de configuração permite estender facilmente o uso do gerador para outras linguagens ou *frameworks*.

3.5 Limitações do FrameWeb

Sendo baseado no meta-modelo do FrameWeb e utilizando modelos criados a partir do *FrameWeb Editor*, o gerador de código herda eventuais limitações presentes nestes. Algumas destas limitações foram endereçadas no contexto deste trabalho, resultando nas

modificações ao meta-modelo descritas na Seção 3.4. Outras foram classificadas como fora de escopo e são descritas nesta seção.

Atualmente no **Modelo de Navegação** são encontradas as seguintes limitações:

- Não oferece o suporte a configuração de atributos dos componentes, i.e., informar uma classe de uma determinada *tag*.
- Não possui suporte para componentes que contém componentes, i.e., componente *dataTable* do JSF não pode ser representado com outros componentes em seu interior. Atualmente apenas formulários podem conter componentes.

No Modelo de **Aplicação**, **Entidade** e **Persistência**:

- Não há suporte para implementar interfaces e/ou estender classes não definidas no modelo.
- Não há recurso para anotações em métodos, atributos ou classes.
- Em interfaces não possui suporte para informar assinatura de métodos; (bug editor)
- Não possibilita marcar um atributo como final.
- Não possibilita criar métodos construtores.

Além destas, limitações no uso do *FrameWeb Editor* por parte de alunos da disciplina de Desenvolvimento Web foram registradas no *issue tracker* do projeto FrameWeb, disponível em <https://github.com/nemo-ufes/FrameWeb/issues>.

Os itens acima podem servir de sugestões para futuras evoluções ou mudanças no método, a fim de torná-lo mais representativo e fiel à necessidade diária no desenvolvimento de software.

4 Avaliação do Trabalho

O propósito do FrameWeb é minimizar o esforço na etapa de desenvolvimento, bastando executá-lo junto ao gerador de código. Para avaliar o resultado de seu uso, foram utilizados dois projetos: um trabalho desenvolvido por alunos de pós-graduação da UFES, na disciplina de Desenvolvimento Web e Web Semântica (DWWS) no ano de 2016 e a aplicação `crud-jsf`, desenvolvida por Adriano Ramos Santos e disponibilizada no *GitHub*.[1]

A métrica de avaliação utilizada se baseia na quantidade de *tags* geradas e a real quantia desenvolvida nas aplicações. Algumas *tags* não entraram nos cálculos, devido sua utilização ser esporádica e de uso particular em uma página HTML, são elas `<br />` e `<hr />`.

Componentes compostos por outros componentes como painéis podem ser modelados no editor gráfico, porém, há necessidade de ajuste manual dos componentes ali contidos. Vale ressaltar que a medida aplicada não determina as configurações ou estado de cada propriedade de uma determinada *tag*, sendo esse passo de responsabilidade do próprio desenvolvedor. Como já descrito nas sessões anteriores, o FrameWeb é um auxílio para o desenvolvimento, principalmente em sua etapa inicial.

Este capítulo, portanto, apresenta os resultados obtidos na aplicação dos softwares descritos no Capítulo 3. A Seção 4.1 apresenta os resultados obtidos na geração de código da aplicação desenvolvida por alunos da disciplina de DWWS. A Seção 4.2 apresenta a avaliação feita no âmbito da aplicação `crud-Jsf`. A Seção 4.3 apresenta um comparativo entre o FrameWeb e os *softwares* desenvolvidos neste documento com outras soluções disponíveis.

4.1 Resultados da aplicação DWWS

O curso de Desenvolvimento Web e Web Semântica (DWWS) é ministrado na Universidade Federal do Espírito Santo (UFES), sua ementa é formada por: conceitos básicos do desenvolvimento Web; desenvolvimento Web na plataforma Java EE; fundamentos da Web Semântica; projetando, publicando e consumindo dados interligados (*linked data*), modelando aplicações Web baseadas em *frameworks* com *FrameWeb*.

Ao final do período, os discentes entregam um projeto desenvolvido com base nas tecnologias apresentadas durante o curso e sua documentação deve ser entregue nos padrões do método *FrameWeb*, este é o artefato utilizado neste trabalho, com isso possibilitando a comparação entre o código entregue e o construído pelo gerador. Um destes projetos foi

[1] https://github.com/SRamosAdriano/crud-jsf

utilizado para avaliação do Gerador FrameWeb: código foi gerado a partir dos modelos construídos pelos alunos e, em seguida, comparado com os artefatos de código desenvolvidos manualmente pelos mesmos alunos.

A Tabela 3 representa a quantidade de *tags* existente em cada arquivo. A coluna TPT/A mostra o total de *tags*/atributos encontrado no arquivo original; a coluna GTP/A contém o total de *tags*/atributos gerados pelo gerador; e, finalmente, a coluna Total T/A exibe o percentual de *tags* e atributos gerados (em comparação com o original), respectivamente. O uso de *tags* como métrica de avaliação foi utilizada pois seu uso compõem projetos com suporte a interface para web.

Arquivo (xhtml)	TPT/A	GTP/A	total T/A(%)
formNovoUsuario	68/162	52/66	76% / 40%
formUsuario	69/154	55/66	79% / 42%
listUsuario	32/77	23/31	71% / 40%

Tabela 3 – Comparativo entre o projeto real e gerado com base no *framework* FrameWeb

Pode-se observar na Tabela 3 um resultado acima de 70% para os arquivos do exemplo relativo a criação das *tags* de forma rápida e simples para o desenvolvedor, e 40% para o total de atributos preenchidos. O resultado baixo para atributos era esperado, devido haver poucas informações no modelo, contudo, já há iniciativa de alteração no metamodelo para adequação de atributos dinâmicos e estáticos no arquivo de perfil de *framework*. Desta forma tanto o Editor FrameWeb quanto o gerador disponibilizarão para o usuário a possibilidade de informar valores aos atributos ou criá-los dinamicamente.

Entre os objetivos futuros está repetir a avaliação relatada na Tabela 3 com outros modelos produzidos por alunos de pós-graduação da universidade de modo a avaliar ainda mais o gerador e identificar possibilidades de evolução dos (meta-)modelos de FrameWeb, acrescentando ainda os demais modelos propostos pelo FrameWeb (Entidade, Persistência, Aplicação), em especial o modelo de Aplicação, por haver informações referentes a um certo domínio ao qual o sistema pertencerá, ou seja, juntamente com o modelo de Navegação são os modelos em que mais se encontram versatilidade no código, tanto em relação a número de linhas quanto em definições de funcionalidades, uma vez que essas características são exclusivas de cada sistema e seu contexto. Ao contrário, os modelos de Entidade e Persistência, em sua maioria, possuem menos variações. Todavia, o FrameWeb tende a abstrair isso para o desenvolvedor, tornando assim menos “trabalhosa” a construção do código fonte.

4.2 Desenvolvimento de Aplicação Utilizando o Gerador de Código (crud-jsf)

Como prova de conceito do funcionamento do gerador de código, foi utilizado o projeto `crud-jsf`,[2] desenvolvido por Adriano Ramos Santos e disponibilizado no GitHub.[3]

O projeto `crud-jsf` consiste em um exemplo de aplicação desenvolvida com os *frameworks* JSF[4] versão 2.0, PrimeFaces[5] e Hibernate[6] e projetado para ser executado com JBoss Enterprise Application Platform 6.[7] A aplicação `crud-jsf` permite o cadastro, listagem, alteração e exclusão de livros e autores. Foi selecionada para ser utilizada como prova de conceito para o gerador de código por se tratar de uma aplicação simples e funcional.

A aplicação oferece recursos para gerenciar os registros de livros, autores e o relacionamento entre eles (quem são os autores de cada livro). Por se tratar de um *CRUD* (sigla para *create*, *retrieve*, *update* e *delete*), oferece as ferramentas necessárias para realizar as operações de cadastro, exibição, alteração e exclusão, tanto dos livros quanto dos autores (e o relacionamento entre eles).

Para a prova de conceito, o primeiro passo foi a criação de um modelo baseado no código fonte do projeto, por meio de engenharia reversa e utilizando o *Editor FrameWeb* foi construído o modelo de Navegação, Entidade, Aplicação e Persistência. Cada modelo será descrito nas subseções a seguir.

4.2.1 Modelo de Navegação

Na Figura 11, temos o diagrama do Modelo de Navegação do projeto `crud-jsf`. É possível observar que há quatro pacotes: `br.com.sramos.crudjsf.controller` na camada de controle, e os outros três na camada de visão: `autor`, `livro` e um pacote sem nome.

Ao efetuar a leitura do modelo referente à camada de controle, o gerador irá transformar o nome do pacote em uma estrutura de diretórios, sendo o delimitador o caractere ponto (`.`). Por exemplo, o primeiro item `br.com.sramos.crudjsf.controller`, teremos como resultado a estrutura da Figura 12. Por ser parte do código-fonte Java da aplicação, essa estrutura será criada dentro do diretório definido pelo usuário na chave de configuração `dir_output_class` (descrita na Tabela 1).

Na parte de visão, temos o pacote sem nome contendo duas páginas: `index.html`

[2] `crud-jsf`, <https://github.com/SRamosAdriano/crud-jsf>
[3] GitHub, <https://github.com/>
[4] JSF, <http://www.oracle.com/technetwork/20java/javaee/javaserverfaces-139869.html>
[5] PrimeFaces, <https://www.primefaces.org/>
[6] Hibernate, <http://hibernate.org/>
[7] JBOSS, <https://www.redhat.com/pt-br/technologies/jboss-middleware/application-platform>

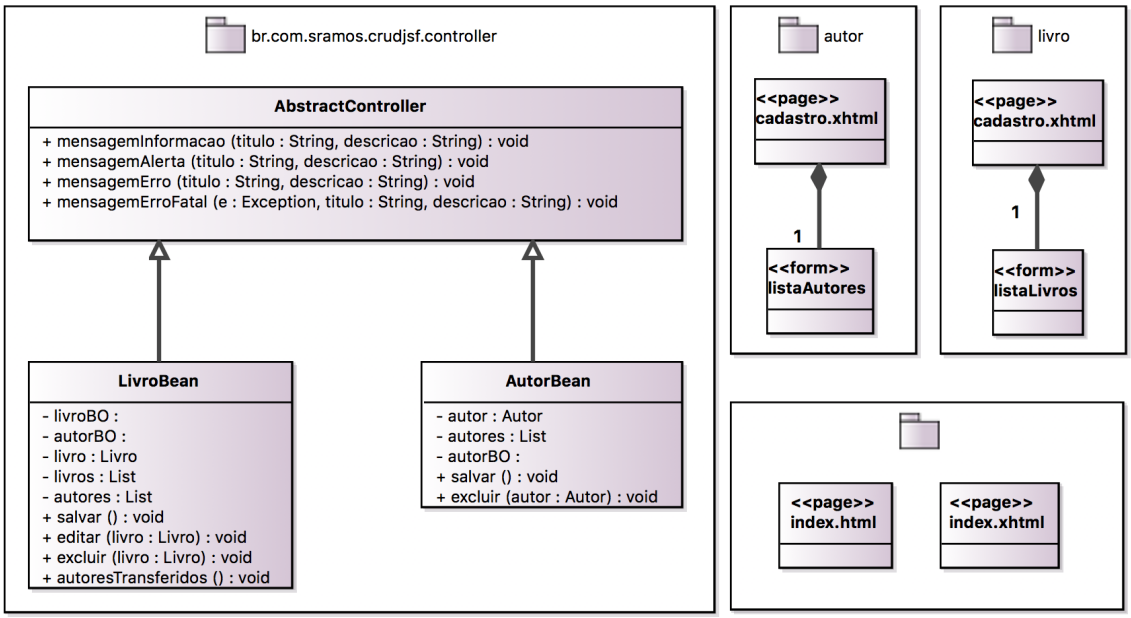

Figura 11 – Modelo de navegação do projeto `crud-jsf`.

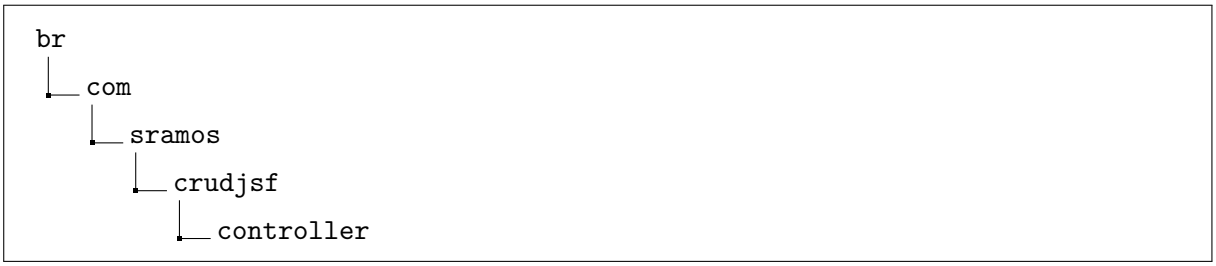

Figura 12 – Estrutura de diretório criada pelo gerador.

e `index.xhtml`. Ao não informar um nome de diretório para o pacote, o gerador irá criar os arquivos contidos diretamente na raiz do diretório definido pela chave de configuração `dir_output_web`, visto que o pacote de visão se refere aos artefatos Web da aplicação. De maneira análoga, os elementos dos pacotes `autor` e `livro` serão gerados em diretórios abaixo de `dir_output_web` de nome idêntico ao nome do pacote.

4.2.2 Modelo de Entidades

Na Figura 13, temos o Modelo de Entidades do projeto `crud-jsf`. Nesse diagrama é possível observar as classes `Livro` e `Autor` seus respectivos atributos. Vale ressaltar que os atributos `id` e `dataNascimento` não são como os demais (instâncias de `DomainAttribute`), pois são instâncias de `IdAttribute` e `DateTimeAttribute`, respectivamente. Embora sejam visualmente semelhantes, internamente, no arquivo onde são gravadas as informações dos modelos, esses atributos possuem tipos (`xsi:type`) diferentes. Tais informações são de alta importância para o gerador, pois o mesmo se baseia em cada tipo ao realizar e/ou processar a transformação do modelo em código, também possibilitando, em alguns casos, a aplicação de regras diferentes a cada tipo de elemento representado no arquivo de modelo.

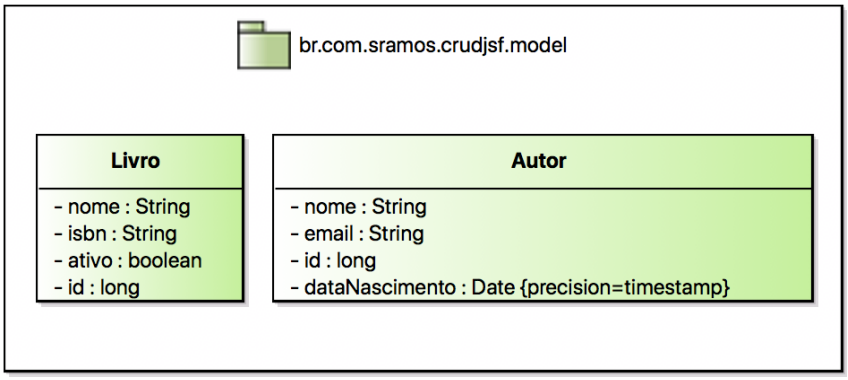

Figura 13 – Modelo de entidade do projeto `crud-jsf`.

Outro ponto importante é a saída dos arquivos construídos, eles serão armazenados dentro do caminho do pacote correspondente a cada classe, nesse exemplo do model, há o pacote `br.com.sramos.crudjsf.model` então gerador irá criar toda a estrutura de diretórios `src/br/com/sramos/crudjsf/model` e gravar os arquivos. Isso ocorre em todos os modelos, os arquivos sempre serão armazenados dentro do seu pacote.

4.2.3 Modelo de Aplicação

Na Figura 14, temos o modelo de aplicação do projeto `crud-jsf`. No diagrama há pacotes, classes e interfaces. Em especial, existe parte do modelo de persistência nesse modelo, de modo a representar a dependência da camada de aplicação com a camada de persistência. Seguindo o método *FrameWeb*, o *Editor FrameWeb* possibilita diretamente na ferramenta a ligação entre o modelo de aplicação e o modelo persistência. No código fonte do projeto `crud-jsf` a classe `AutorBOImpl` possui um atributo da interface `AutorDAO`, representado no modelo pela ligação entre a classe `AutorBOImpl` e a interface `AutorDAO` do modelo de persistência. O gerador será o responsável por processar essa ligação entre os elementos do diagrama e criar a propriedade na classe `AutorBOImpl` durante o processo de transformação do modelo.

As classes `AutorConverter` e `LivroConverter` estendem uma classe conversor do *JavaServer Faces* que possibilita a converter strings em objetos e objetos em strings, conforme necessário. Sua utilização é necessária quando os conversores padrão incluídos no *JavaServer Faces* não podem executar a conversão de dados necessária, sendo possível, então, criar um conversor personalizado para executar essa conversão especializada. Essa necessidade foi encontrada nesse projeto e sua implementação foi demostrada na Figura 14.

A classe `BusinessException` é responsável por intermediar e/ou organizar as exceções lançadas pelo sistema. A exceção é uma condição de erro ou comportamento inesperado que ocorre durante a execução de um programa afetando o fluxo normal da execução das instruções. Sendo assim, o uso de uma exceção própria do sistema pode ser atribuído a um melhor controle para detecção de possíveis falhas e execução de respostas

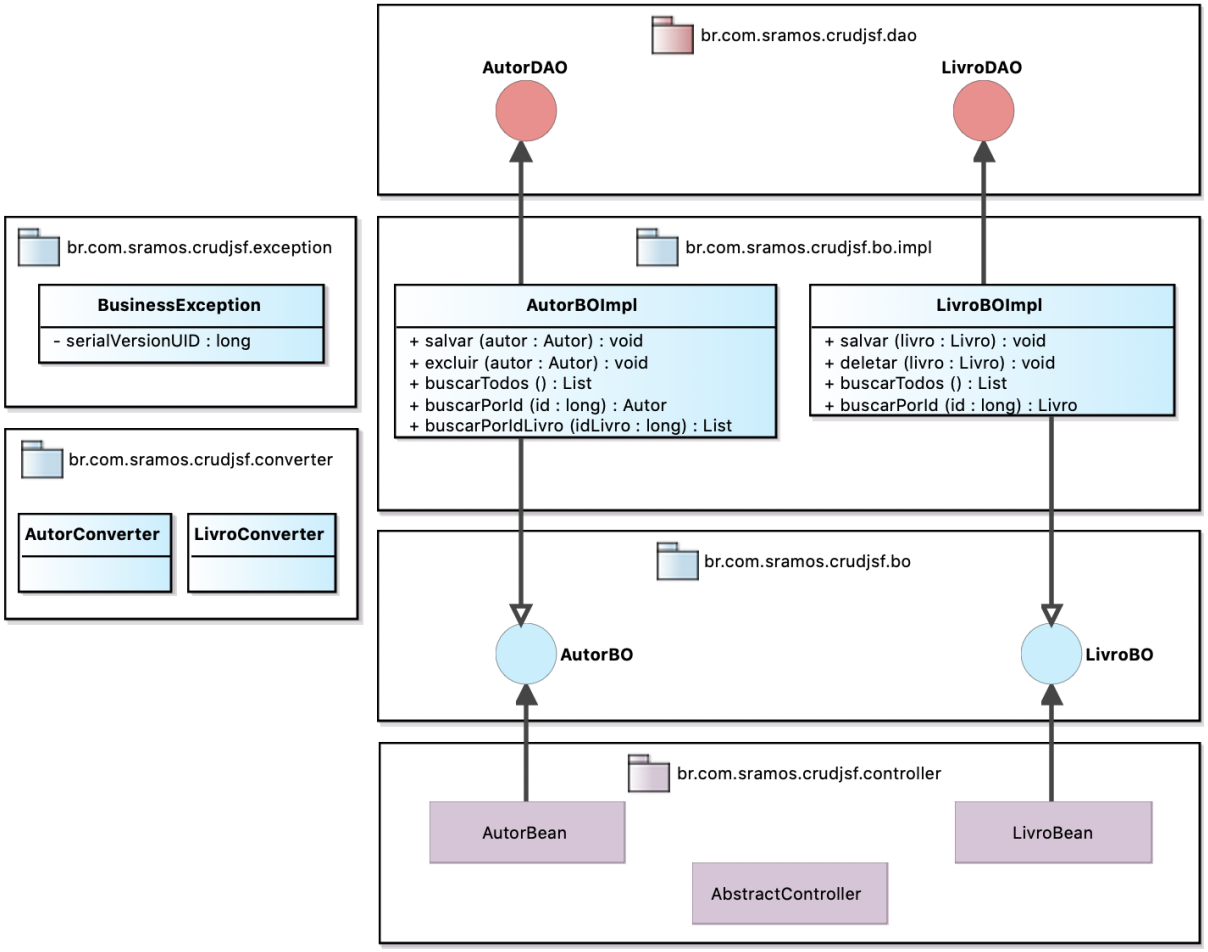

Figura 14 – Modelo de aplicação do projeto `crud-jsf`.

pelo próprio sistema ao detectá-las.

4.2.4 Modelo de Persistência

A Figura 15 mostra as interfaces `LivroDAO` e `AutorDAO`, pertencentes ao pacote `br.com.sramos.crudjsf.dao`. Elas são responsáveis pela persistência dos atributos no banco de dados. No pacote `br.com.sramos.crudjsf.dao.impl` temos as classes `LivroJPA` e `AutorJPA`, que estendem tanto da classe `AbstractJPA` e implementam suas respectivas interfaces DAO. Por se tratar de um CRUD simples, os DAOs não possuem nenhum método de consulta específica, apenas os métodos de persistências básicos, implementados na classe `AbstractJPA`.

4.2.5 Resultados

Após o processamento de todas as etapas descritas anteriormente, o gerador terá criado resultado o código fonte da aplicação proposta. Na Figura 16 é possível observar os arquivos e diretórios que foram gerados, nesse caso houve uma cobertura de 100% com o projeto deste experimento `curdjsf`, em termos de arquivos e diretórios gerados.

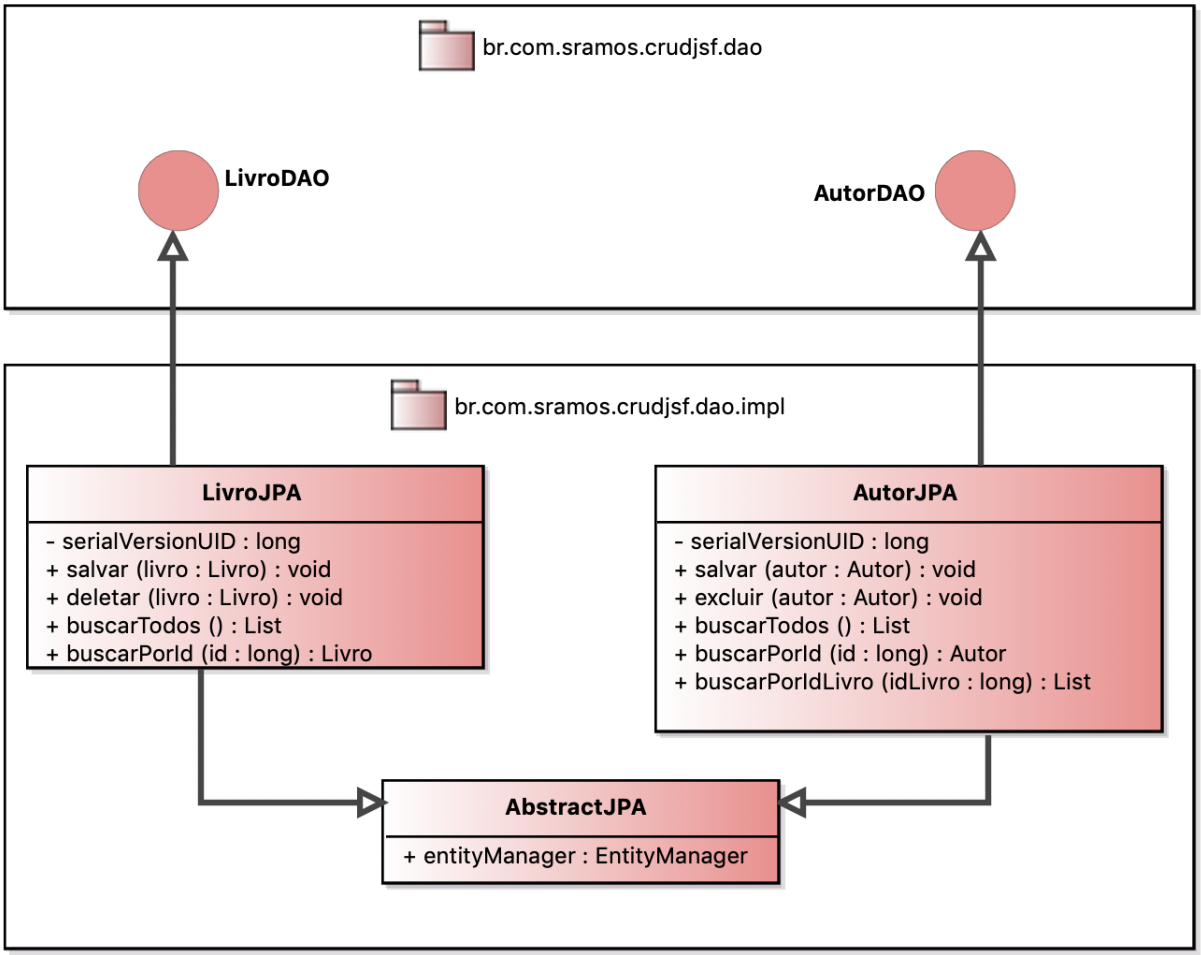

Figura 15 – Modelo de persistência do projeto `crud-jsf`.

Em relação ao conteúdo dos arquivos, na Figura 17, é apresentada uma comparação entre a classe `AutorBean` original e a construída pelo gerador. Para simplificar a imagem foi removido o conteúdo dos métodos `salvar, excluir e getAutores` do arquivo original, que se tratavam de coisas específicas dos métodos (e que devem, necessariamente, serem escritas pelo desenvolvedor). É possível observar que o arquivo gerado há mais métodos do que seu arquivo espelho, isso ocorre pelo fato de que o processamento de um modelo seguirá sempre o padrão da criação de *get* e *set* para todas as propriedades. Outro ponto importante é a não cobertura da implementação da interface *Serializable* e a propriedade *serialVersionUID*, isso se deve a limitações atuais do metamodelo *FrameWeb*. Sendo assim necessário o ajuste por parte do desenvolvedor.

4.3 Trabalhos Relacionados

Semelhante à proposta deste documento, existe o projeto MERLIN (MRACK, 2009), focado na geração de interfaces para CRUD. Suas principais funcionalidades são: (i) a geração das telas em tempo de execução; (ii) a configuração norteada pela edição textual

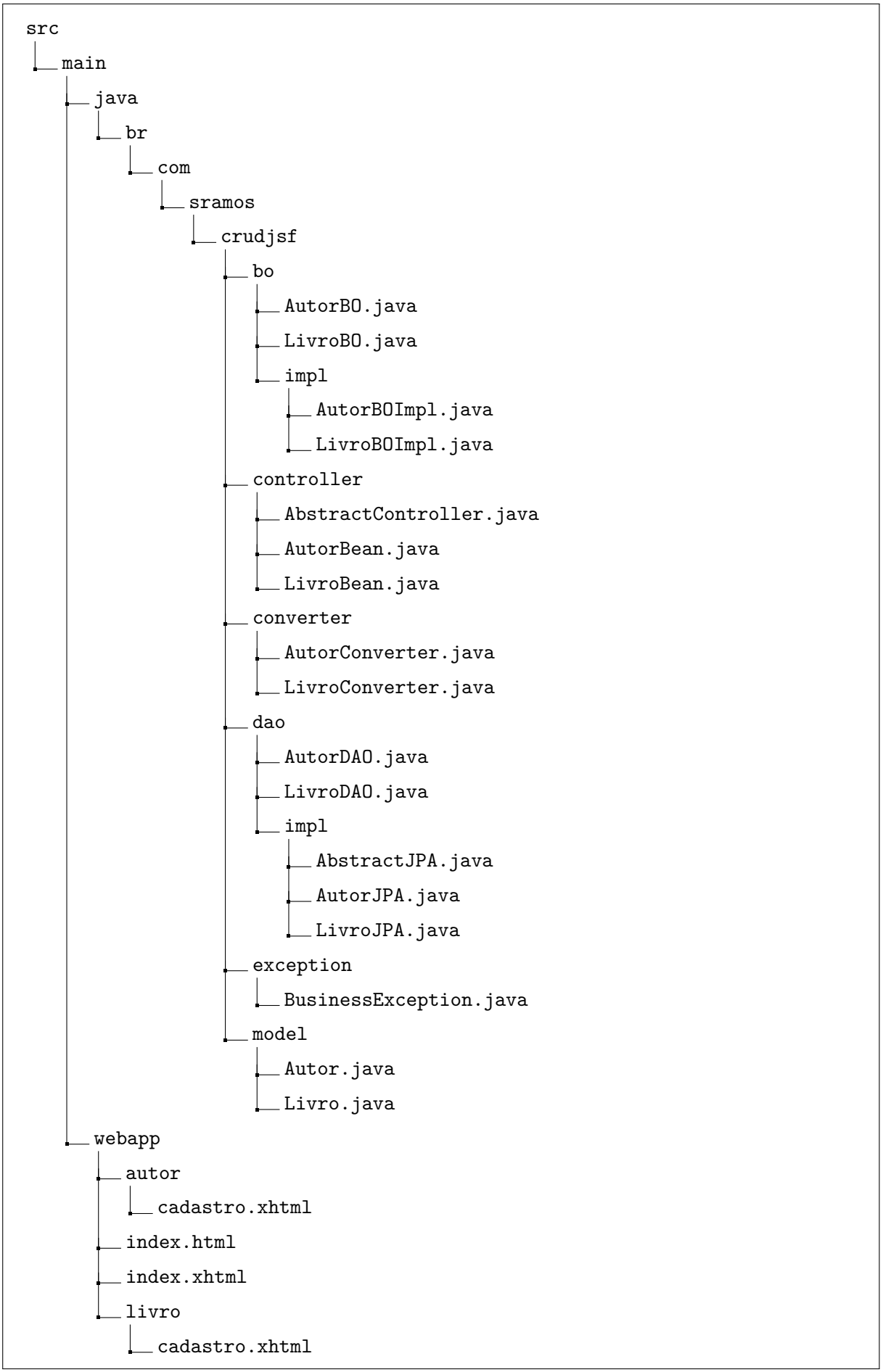

Figura 16 – Estrutura de diretório do projeto `crud-jsf`.

Original

```
@ManagedBean(name="autorBean")
@ViewScoped
public class AutorBean extends AbstractController
implements Serializable{
    private Autor autor = new Autor();
    private List<Autor> autores;
    @EJB
    AutorBO autorBO;

    public void salvar(){}
    public void excluir(Autor autor){}

    public Autor getAutor() {return autor;}
    public void setAutor(Autor autor) {
        this.autor = autor;
    }

    public List<Autor> getAutores() {
        return autores;
    }

    private static final long serialVersionUID =
-3083386675935212 78L;
}
```

Generator

```
@ManagedBean(name = "autorBean")
@ViewScoped
public class AutorBean extends AbstractController {
    private Autor autor;
    private List autores;
    private AutorBO autorBO;

    public void salvar(){}
    public void excluir(Autor autor){}


    public Autor getAutor(){return autor;}
    public void setAutor(Autor _autor){
        autor = _autor;
    }

    public List getAutores(){
        return autores;
    }

    public void setAutores(List _autores){
        autores = _autores;
    }

    public AutorBO getAutorBO(){
        return autorBO;
    }

    public void setAutorBO(AutorBO _autorBO){
        autorBO = _autorBO;
    }
    public AutorBean(){}
}
```

Figura 17 – Comparação entre a classe original e a construída pelo gerador `crud-jsf`.

assistida dos modelos; (iii) uma base construída em padrões e linguagens de mercado; (iv) o reuso transparente e gerenciado de configurações e (v) a modelagem centrada no Modelo de Domínio da aplicação. Embora semelhante, o FrameWeb possui sua estrutura voltada para sistemas Web e criação de interfaces mais dinâmicas, além do CRUD, possibilitando ao desenvolvedor maior controle sobre os componentes de interface a serem gerados, e tudo isso diretamente na criação do modelo. Além disso, o gerador cria o código fonte, possibilitando ao programador atualizar/aprimorar sua interface, conforme sua necessidade.

O MVCASE (PRADO; LUCRéDIO, 2001) é uma abordagem orientada a objetos para o desenvolvimento de *software* que utiliza o método Catalysis e gera seu código para uma plataforma distribuída por meio da tecnologia *Enterprise Java Beans* (EJB). Ao utilizar POO, o MVCASE possibilita a reutilização dos modelos criados, assim como o Editor FrameWeb, que também possibilita a reutilização dos modelos construídos. Desta forma, otimiza-se o tempo de desenvolvimento, já que não há necessidade de reconstruir etapas semelhantes de projetos. A principal vantagem do metamodelo FrameWeb sobre o MVCASE é sua possibilidade de extensão, i.e., ao final pode-se ter como resultado um código em diferentes linguagens e *frameworks*, desde que definidos previamente.

O GeCA (STEINMACHER et al., 2006) propõe-se a permitir aos analistas e desenvolvedores realizarem engenharia reversa e executar a manutenção dos sistemas de maneira facilitada. Após a realização do processo de Engenharia Reversa, o sistema pode ser novamente gerado nos formatos suportados pelo gerador do GeCA. Embora o

FrameWeb não ofereça suporte até o momento para Engenharia Reversa de código (essa funcionalidade será abordada em trabalhos futuros como evoluções do *framework*), ele oferece suporte para geração código a partir de modelo, e como nessa etapa são utilizado modelos de arquivos (*template*), possibilita ao desenvolvedor editar esses arquivos para que, no ato de geração, o resultado final seja conforme sua necessidade ou padrão desejado (*design patterns*).

GenERTiCA (WEHRMEISTER et al., 2008) é uma ferramenta de geração de código que utiliza como entrada um modelo UML e conjunto de regras de mapeamento (descrito através de um arquivo XML). Tem por finalidade gerar o código mais completo possível (não apenas esqueletos de classes como a maioria das ferramentas faz) sem restrição. O trabalho aborda o problema da geração automática de código de sistemas DERTS (*Distributed Embedded Real-Time Systems*), com foco nos requisitos funcionais e requisitos não funcionais, usando conceitos de paradigma de programação orientada a aspectos. Mesmo o FrameWeb não tendo como proposta direta a abordagem de requisitos funcionais e não funcionais, eles são apresentados nos modelos, pois são parte fundamental para qualquer sistema, sendo assim, o FrameWeb com o seu editor possibilita de forma mais simples e fácil a sua declaração nos modelos que são criados.

O WebML (BONGIO et al., 2000) é uma notação para especificar visualmente *sites* complexos no nível conceitual, incluindo entrada de dados e unidades de operação. O WebML também é baseado em XML e permite a geração automática de código HTML. Em comparação com o FrameWeb, suas interfaces são mais precisas, pois seu editor provê detalhes sobre organização dos componentes na página, diferentemente do FrameWeb Editor que recebe as informações de quais componentes devem compor a página e não sua organização. Porém, o FrameWeb, além das páginas Web, também pode gerar os códigos necessários para os eventuais processos realizados pela interface, processo ainda não suportado pelo WebML.

5 Considerações Finais

Nesta seção encontram-se as conclusões sobre os resultados obtidos e as propostas de melhorias para versões futuras.

O objetivo deste trabalho foi apresentar uma solução para geração de código com base no método FrameWeb e contribuir com a evolução de seu metamodelo. A meta principal deste trabalho é motivar a criação de sistemas baseados em modelos, auxiliando e simplificando algumas etapas do desenvolvimento de um sistema Web e seu reaproveitamento de forma simples e fácil.

Durante o desenvolvimento do gerador, foram necessárias algumas alterações no metamodelo do FrameWeb, descritas na Seção 3.4. Tais alterações representam uma contribuição significativa não só para este, mas também para outros projetos similares que venham a ser desenvolvidos, uma vez que permitiram uma maior integração entre o FrameWeb e o gerador de código. O Gerador FrameWeb mostrou-se eficiente na construção da estrutura de diretórios e na geração das *tags* em ambos os projetos.

Esse gerador está em constante evolução, a fim de atingir a completude do FrameWeb. O demostrado neste trabalho é uma porção das facilidades previstas em sua utilização. Futuramente seu uso será integrado com o FrameWeb Editor, para facilitar a construção diretamente pelo editor. Atualmente, pode ser usado via linha de comando, via aplicativo *desktop* simples, ou ainda via interface Web, pelas quais o usuário seleciona um arquivo e o aplicativo realiza a chamada do executável passando o arquivo selecionado como parâmetro, desta forma, facilitando o uso por usuários leigos.

O gerador juntamente com o editor serão levados à disciplina de graduação Desenvolvimento Web e Web Semântica (DWWS) na Universidade Federal do Espírito Santo, para facilitar a criação dos trabalhos propostos aos alunos durante o semestre letivo. Uma experiência já foi realizada no semestre 2018/2, porém os resultados não foram analisados a tempo de serem incluídos neste documento.

O metamodelo FrameWeb está constantemente sendo revisado e, com isso, gerando evoluções. Algumas serão propostas em dissertações ou novos artigos, tais como: (i) criação de CRUD de forma automática para entidades definidas no modelo; (ii) adicionar suporte para *Web API*; (iii) adicionar suporte para criação de código voltado a Web Semântica, com base em anotações realizadas no modelo de entidades propostas por Celino et al. (2016); (iv) geração de código para dispositivos móveis por meio de plataformas híbridas e/ou nativas, dentre outras.

De forma mais abrangente, este trabalho contribui na evolução do *FrameWeb*, nos

quesitos mais específicos e inerentes ao projeto, como testes, validações, integrações e alterações de seus modelos, a fim de confirmar o seu uso e construir um novo caminho para a sua evolução e implementações futuras. Outra contribuição deste trabalho é a metodologia aplicada no desenvolvimento do gerador, na etapa de construção do aplicativo se prezou o uso de meios simples e efetivos para realização das tarefas necessárias para transformação de código, de modo que sua estrutura pode ser reutilizada em outros projetos semelhantes ou adição de novas características por meio de implementação das interfaces disponíveis no código e explicadas neste trabalho.

Vale ressaltar que há inúmeras implementações que podem ser acrecidas ao gerador para melhorar ainda mais o benefício de seu uso, e.g., interface visual para execução de comandos, edição e adição de novas linguagens de programação e frameworks. Por fim, seu uso para contribuir com desenvolvedores e/ou pesquisadores que possam necessitar de alguma base ou fonte de estudo para projetos baseados em modelos.

Referências

ALMEIDA, N. Vittorazzi de; CAMPOS, S.; SOUZA, V. S. A model-driven approach for code generation for web-based information systems built with frameworks. In: . [S.l.: s.n.], 2017. p. 245–252. Citado na página 25.

ALUR, D.; CRUPI, J.; MALKS, D. *Core J2EE Patterns: Best Practices and Design Strategies*. 2nd. ed. [S.l.]: Prentice Hall / Sun Microsystems Press, 2003. Citado na página 34.

BONGIO, A. et al. Web Modeling Language (WebML): a modeling language for designing Web sites. 2000. Citado na página 62.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML: guia do usuário*. [S.l.]: Elsevier Brasil, 2006. Citado 3 vezes nas páginas 22, 32 e 34.

CAMPOS, S. L.; SOUZA, V. E. S. FrameWeb Editor: Uma Ferramenta CASE para suporte ao Método FrameWeb. SBC, Gramado, RS, Brazil, p. 199–203, oct 2017. Citado 3 vezes nas páginas 23, 25 e 37.

CELINO, D. R. et al. 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*. Teresina, PI, Brazil: ACM, 2016. p. 231–238. Citado 2 vezes nas páginas 25 e 63.

CONALLEN, J. *Desenvolvendo aplicações Web com UML: tradução da segunda edição*. ELSEVIER EDITORA, 2003. ISBN 9788535212099. Disponível em: <https://books.google.com.br/books?id=gds8AAAACAAJ>. Citado na página 42.

CORRÊA, C. K. F. *ODYSSEY-MEC: UMA ABORDAGEM PARA O CONTROLE DA EVOLUÇÃO DE MODELOS COMPUTACIONAIS NO CONTEXTO DO DESENVOLVIMENTO DIRIGIDO POR MODELOS*. Tese (Doutorado) — Universidade Federal do Rio de Janeiro, 2009. Citado na página 31.

FOWLER, M. *UML Essencial: um breve guia para linguagem padrão*. [S.l.]: Bookman Editora, 2014. Citado na página 22.

FRAKES, W. B.; KANG, K. Software reuse research: Status and future. *IEEE Transactions on Software Engineering*, IEEE, v. 31, n. 7, p. 529–536, 2005. ISSN 00985589. Citado na página 22.

ISAKOWITZ, T.; BIEBER, M.; VITALI, F. Web information systems. *Communications of the ACM*, ACM, v. 41, n. 7, p. 78–80, 1998. Citado na página 21.

JOHNSON, R. et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995. ISBN 9780201633610. Disponível em: <https://books.google.com.br/books?id=iyIvGGp2550C>. Citado na página 29.

MALDONADO, J. C. et al. Padrões e frameworks de software. *Notas Didáticas, Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, ICMC/USP, São Paulo, SP, Brasil*, 2002. Citado na página 28.

MARTINS, B. F. *Evolução do Método FrameWeb para o Projeto de Sistemas de Informação Web Utilizando uma Abordagem Dirigida a Modelos*. [S.l.], 2016. Citado 3 vezes nas páginas 13, 23 e 36.

MARTINS, B. F.; SOUZA, V. E. S. 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*. [S.l.]: ACM, 2015. p. 41–48. Citado 4 vezes nas páginas 13, 25, 34 e 36.

MEDEIROS, E. R. d. Novastudio: gerador de código usando a arquitetura dirigida pelos modelos (mda). 2009. Citado 3 vezes nas páginas 30, 31 e 41.

MINETTO, E. L. Frameworks para desenvolvimento em php. *São Paulo: Novatec*, 2007. Citado na página 28.

MRACK, M. *Geração Automática e Assistida de Interfaces de Usuário*. [S.l.], 2009. Citado na página 59.

PASTOR, O. et al. Model-driven development. *Informatik-Spektrum*, v. 31, p. 394–407, 2008. Citado na página 30.

PERUCH, L. A. Aplicação e análise do método frameweb com diferentes frameworks web. 2007. Citado na página 29.

PRADO, A. F.; LUCRéDIO, D. Ferramenta MVCASE - Estágio Atual: Especificação, Projeto e Construção de Componentes. 2001. Citado na página 61.

PRESSMAN, R. S. *Software engineering: a practitioner's approach*. [S.l.]: Palgrave Macmillan, 2005. Citado na página 27.

SILVA, R. P. e. Suporte ao desenvolvimento e uso de frameworks e componentes. Instituto de Informática da Universidade Federal do Rio Grande do Sul., 2000. Citado na página 21.

SOMMERVILLE, I. *Software Engineering*. 9th. ed. USA: Addison-Wesley Publishing Company, 2010. ISBN 0137035152, 9780137035151. Citado na página 22.

SOUZA, B. F. M. Evolução do método frameweb para o projeto de sistemas de informação web utilizando uma abordagem dirigida a modelos. Universidade Federal do Espírito Santo, 2016. Citado na página 31.

SOUZA, V. E. S. *FrameWeb: um Método baseado em Frameworks para o Projeto de Sistemas de Informação Web*. [S.l.], 2007. Disponível em: <http://portais.ufes.br/PRPPG/ext/mono.php?progpess=2032&curso=9&prog=30001013007P0>. Citado 7 vezes nas páginas 21, 22, 23, 25, 27, 28 e 34.

STEINMACHER, I. et al. GeCA: Uma Ferramenta de Engenharia Reversa e Geração Automática de Código. 2006. Citado na página 61.

WEHRMEISTER, M. A. et al. GenERTiCA: A Tool for Code Generation and Aspects Weaving. 2008. Citado na página 62.

Apêndices