ISSN: 1984-3151

GERADOR DE CÓDIGOS PARA O DESENVOLVIMENTO DE APLICAÇÕES WEB A PARTIR DA MODELAGEM ENTIDADE-RELACIONAMENTO

CODE GENERATOR FOR WEB APPLICATIONS DEVELOPMENT FROM ENTITY-RELATIONSHIP MODEL

Cristiano Martins Monteiro[1]; Flavianne Braga Campos de Lima[2]; Carlos Renato Storck[3]

1 Bacharel em Ciência da Computação. Centro Universitário de Belo Horizonte - UniBH, 2013. Aluno de Mestrado do Centro Federal de Educação Tecnológica de Minas Gerais – CEFET-MG. Belo Horizonte, MG. cristianomartinsm@gmail.com.

2 Bacharel em Ciência da Computação. Centro Universitário de Belo Horizonte - UniBH, 2013. Belo Horizonte, MG. flavianne.lima@hotmail.com.

3 Mestre em Informática. Pontifícia Universidade Católica de Minas Gerais – PUC-MG, 2007. Professor do Centro Federal de Educação Tecnológica de Minas Gerais. Contagem, MG. storck@contagem.cefetmg.br.

Recebido em: 20/09/2015 - Aprovado em: 20/05/2016 - Disponibilizado em: 31/05/2016

RESUMO: A geração automática de código-fonte é uma prática adotada no desenvolvimento de softwares para agilizar, facilitar e padronizar a implementação dos projetos. Embora seja uma prática comum nas fábricas de software, não se conhece uma ferramenta que permita escolher o padrão de projeto a ser usado. O objetivo principal deste trabalho é apresentar um gerador de códigos para o desenvolvimento de sistemas Web a partir de uma modelagem entidade-relacionamento, uma linguagem de programação e um padrão de projeto determinados pelo usuário. Os objetivos específicos são propor uma arquitetura do sistema capaz de adequar e reaproveitar diferentes padrões de projeto, linguagens de programação e projetos cadastrados; permitir que o usuário cadastre, altere, exclua, importe e exporte um projeto; e gerar automaticamente o seu código-fonte e scripts de banco de dados. Este trabalho se justifica pela importância de reduzir erros de codificação; e evitar perca tempo ao realizar atividades rotineiras de implementação de padrões de projeto. Possibilitando assim, maior dedicação no planejamento das regras de negócios e redução de custos. A ferramenta proposta (GCER) foi desenvolvida em linguagem Java com o uso banco de dados Oracle 11g, e seguindo os padrões DAO e MVC. Os resultados foram avaliados através da geração e compilação de códigos de um projeto para cadastro de veículos. A geração com êxito evidencia a viabilidade da ferramenta proposta para a geração automática de códigos no processo de desenvolvimento de software.
PALAVRAS-CHAVE: Gerador Automático de Código. Padrões de Projeto. Modelagem Entidade-Relacionamento.

ABSTRACT: The automatic generation of source code is a practice adopted in the development of software to streamline, facilitate and standardize the implementation of projects. Although it be a common practice in software factories, it is not known a tool able to choose the design pattern to be used. The main objective of this paper is to present a code generator for the development of Web systems from an entity-relationship modeling, a programming language and a design pattern determined by the user. The specific objectives are to propose a system architecture able to suit and reuse different design patterns, programming languages and saved projects; allow the user to insert, update, delete, import and export a project; and automatically generate the source code and database scripts. This work is justified by the importance to reduce errors of coding; and to avoid waste of time in the development of Web systems performing routine tasks. Allowing, then, a greater dedication in the planning of business rules and the reduction of costs. The tool proposed (GCER) was developed in Java with the database

using Oracle 11g, and following the DAO and MVC patterns. The results were evaluated by generating and compiling codes of a project for vehicle registration. The successful code generation demonstrate the feasibility of the proposed tool for the automatic generation of code in the software development process.
KEYWORDS: Automatic Code Generator. Design Pattern. Entity-Relationship Modeling.

---

1 INTRODUÇÃO

O avanço dos recursos computacionais e disseminação da Internet contribuíram para o aumento da demanda das empresas por sistemas Web (COELHO, 2006).

Esse aumento da procura por aplicações Web fez com que sejam utilizadas ferramentas para a geração de código automático durante a criação do *software*. Assim, evita-se a perda de tempo para a implementação de atividades rotineiras (como o desenvolvimento de telas de cadastro básico), e facilita a integração de tarefas diferentes mas correlacionadas dentro do mesmo sistema (como a programação e a definição do *layout)*.

Rosário, Grande e Pazin (2011) também apontam, como um grande benefício, a utilização de geradores de código, possibilitando o reuso de um artefato já desenvolvido e testado previamente. Dessa forma, diminui-se a possibilidade de erros nos códigos, aumentando a qualidade do produto final.

Além disso, a geração de código automática padroniza as implementações, facilitando as possíveis futuras alterações no *software*. Essas alterações podem ser, por exemplo, atualização de tecnologias e mudanças nos requisitos do cliente.

Os requisitos do cliente são fundamentais para o desenvolvimento de um sistema, visto que eles guiarão todas características e funções a serem implementadas. A partir dos requisitos levantados para o *software*, deve-se definir a modelagem de dados do sistema. Segundo Araújo (2008), a modelagem de dados é uma representação simples e abstrata de um sistema real, a qual é fundamental para obter *softwares* com maior qualidade e segurança. Existem diversas técnicas para a modelagem de dados, dentre elas está a mais utilizada que é o modelo entidade-relacionamento.

Esse modelo representa a estrutura lógica do banco de dados, abordando em forma de objetos chamados entidades e pelos relacionamentos entres estes objetos (ARAÚJO, 2008). Geradores de código são ferramentas que originam códigos-fonte de acordo com essas informações fornecidas pelo modelo de dados (MOREIRA; MRACK, 2003).

Coelho (2006) propõe um gerador de códigos HTML em tempo de execução, com validador dos formulários Web e geração de *scripts* SQL através do dicionário de dados armazenado.

Um trabalho similar é apresentado em Rosário, Grande e Pazin (2011). Os autores apresentam um gerador de códigos denominado “AlphaWEB” com as mesmas características que o proposto em Coelho (2006), porém gerando classes Java *Beans* e abrindo mão do validador de formulários Web.

Adamatti (2006) desenvolveu um sistema Web seguindo *frameworks* MVC e utilizando o gerador de códigos “*Middlegen*” para gerar classes em Java a partir da base de dados.

Globalcode (2009) abordou um gerador de aplicações em diferentes arquiteturas (como JSF, Spring e EJB) capazes de inserir, buscar, atualizar e excluir dados.

Oliveira e Mählmann (2010) apresentam um gerador de códigos .NET que automatiza a criação da camada de negócios de um sistema projetado em três camadas e se baseia em *templates* pré-definidos para a geração das classes.

O presente artigo tem como objetivo principal apresentar um gerador de códigos para o desenvolvimento de um sistema Web a partir da modelagem entidade-relacionamento, linguagem de programação e padrões de projeto que o usuário determinar. A ferramenta proposta foi denominada Gerador de Códigos para Entidade-Relacionamento (GCER) e se diferencia dos trabalhos relacionados mencionados anteriormente por permitir a definição do padrão de projeto a ser utilizado.

Os objetivos específicos são propor uma arquitetura do sistema capaz de adequar, adicionar e reaproveitar diferentes padrões de projeto, linguagens de programação e projetos cadastrados; permitir que o usuário cadastre, altere, exclua, importe e exporte um projeto; e gerar automaticamente o seu código-fonte e *scripts* de banco de dados.

O desenvolvimento deste trabalho se justifica pela viabilidade existente em criar uma ferramenta a ser utilizada no processo de desenvolvimento de *software* guiados por padrões de projeto, a fim de gerar produtos com maior agilidade, qualidade e facilidade de manutenção.

Este artigo foi desenvolvido com base em pesquisa em livros, e artigos sobre modelagem de banco de dados, compiladores e geração automática de códigos e artefatos de sistemas Web. O sistema foi implementado em Java, utilizando os *frameworks* JavaServer Faces 2.0 e jQuery 1.9.1, e o banco de dados utilizado foi o Oracle 11g. O método utilizado para a avaliação dos resultados foi a compilação ou interpretação dos códigos e *scripts* gerados para garantir a inexistência de erros das implementações, assim como o feito em Coelho (2006), Oliveira e Mählmann (2010) e Rosário, Grande e Pazin (2011).

A praticidade em utilizar e segurança da ferramenta proposta evidenciam a viabilidade da ferramenta proposta para a geração automática de códigos no processo de desenvolvimento de *software*.

2 FUNDAMENTAÇÃO TEÓRICA

Essa seção aborda conceitos sobre a geração de códigos e ferramentas utilizadas para o desenvolvimento do GCER. A seção 2.1 aborda o conceito, vantagens e desvantagens da geração automática de códigos. A seção 2.2 define o conceito e funções da geração automática de artefatos. A seção 2.3 define modelagem entidade-relacionamento. E a seção 2.4 apresenta as ferramentas Web utilizadas no desenvolvimento do GCER.

2.1 GERAÇÃO AUTOMÁTICA DE CÓDIGOS

Segundo Moreira e Mrack (2003), gerador automático de códigos é uma ferramenta que gera códigos-fontes com o objetivo de facilitar e acelerar os processos de desenvolvimento de sistemas.

Castro (2010) aponta que um gerador automático de códigos utiliza como dados de entrada a base de dados e *templates* dos códigos a serem gerados. Uma base de dados é um arquivo que possui todas as informações relevantes para a geração automática de códigos. Já os *templates* determinam como a saída do programa (código-fonte) será exibido.

De acordo com Oliveira e Mählmann (2010), as vantagens de utilizar uma ferramenta que gera códigos são: a padronização de códigos diminuindo o número de possíveis erros ocorridos devido às diferentes maneiras de implementação; possibilidade de redução de custos do projeto; garantia da aceleração dos processos de desenvolvimento; segurança; e abstração da interação com o modelo de dados.

Já as desvantagens da utilização de geradores de códigos automáticos são: o código-fonte gerado pode não estar em concordância com o protótipo de codificação da equipe de desenvolvimento; e o código-fonte criado pode desconsiderar questões de

desempenho, otimização, estrutura, integração com outros sistemas ou documentação (MOREIRA; MRACK, 2003).

2.2 Geração Automática de Artefatos de *Software*

Segundo Franca (2000), geradores automáticos de artefatos são ferramentas que criam artefatos, podem ser utilizados para gerar códigos, documentação, diagramas, figuras, e casos de testes.

Artefato é um produto gerado como elemento da definição, manutenção ou utilização dos processos de desenvolvimento de software. Um gerador de artefatos deve realizar as seguintes funções: especificar o artefato, fornecendo todas as informações relevantes para a sua criação; gerar o artefato, utilizando informações da sua própria especificação; e, se o artefato gerado for um código, é preciso compilá-lo (ROSÁRIO, GRANDE, PAZIN, 2011).

A arquitetura de um gerador de artefato possui dois elementos fundamentais: o analisador de especificação e o gerador de artefato. O analisador de especificação é responsável pela tradução do conteúdo descrito de uma linguagem para outra, sendo responsável pela implementação das funções de um analisador léxico. O elemento gerador de artefato converte a especificação em uma representação do artefato-alvo, ou seja, o artefato final (FRANCA, 2000).

2.3 Modelagem Entidade-Relacionamento

O modelo entidade-relacionamento é uma representação de um problema a ser modelado. Por ser simples e ocultar detalhes de como os dados são armazenados, torna-se de fácil entendimento aos usuários (ARAÚJO, 2008).

De acordo com Takai, Italiano e Ferreira (2005), a modelagem ocorre a partir das entidades que são objetos contidos no mundo real. Esses objetos possuem propriedades específicas, denominadas atributos, as quais são relevantes para um determinado problema. Quando uma entidade fornece informações para outra, existe uma ligação entre elas chamada de relacionamento.

Os mesmos autores ainda garantem que o modelo entidade-relacionamento possui como objetivos adquirir todas as informações necessárias de um determinado sistema; evitar a redundância de informações; e facilitar o projeto de banco de dados.

2.4 Ferramentas Web Utilizadas no Desenvolvimento do GCER

Dentre as ferramentas utilizadas recentemente no desenvolvimento Web para facilitar e agilizar as implementações estão o JavaServer Faces (JSF) e o jQuery.

Conforme Nunes e Magalhães (2011, *apud* FRANCO, 2011), o JSF é um *framework* baseado em componentes para o desenvolvimento da camada de apresentação de sistemas implementados em Java. Os principais benefícios do uso do JSF estão em maximizar a produtividade e a integração com outras tecnologias Web, e minimizar a complexidade na manutenção do código.

Silva (2010, *apud* ARIATI, 2011) aponta que o jQuery é uma biblioteca desenvolvida em JavaScript compatível com os diversos sistemas operacionais e navegadores e com suporte à CSS e Ajax. Dentre os seus benefícios estão o devido funcionamento indiferente de navegador, facilidade na identificação de *tags* do documento HTML, melhor manipulação de eventos e desenvolvimento de animações.

3 METODOLOGIA

O sistema Web abordado foi desenvolvido em Java e utiliza os frameworks JSF versão 2.2 e jQuery versão 1.9.1.

Durante a implementação do GCER, o JSF foi utilizado na construção das páginas e validação dos campos no servidor. Já o jQuery foi útil para facilitar a interação com o usuário, tornar os formulários mais elegantes e alterar algumas características do *layout,* quando necessário. O jQuery foi utilizado juntamente com o *plugin* jQuery UI versão 1.10. Esse *plugin* permite que seja adicionado ao projeto um tema para os componentes da interface gráfica, além de possibilitar a utilização de alguns recursos de interface, como a criação de formulários em uma janela de diálogo.

Os resultados foram avaliados através da compilação das classes Java geradas, assim como interpretação dos *scripts* SQL e formulários Web gerados. Esses métodos foram úteis para garantir que os códigos gerados não possuem erros, uma das principais vantagens dos geradores automáticos de códigos.

Os modelos de entidade-relacionamento e de classes do GCER são abordados na seção a seguir.

4 DESENVOLVIMENTO DO GCER

As seções 4.1 Modelagem Entidade-Relacionamento e 4.2 Modelo de Classes apresentam, respectivamente, os diagramas das modelagens entidade-relacionamento e de classes do projeto.

Esses diagramas foram planejados visando as funções de cadastro, alteração e exclusão de projetos pelos usuários do sistema. Cada projeto possui entidades (o equivalente a classes ou tabelas do banco de dados), e cada entidade possui atributos (equivalente aos atributos das classes ou colunas das tabelas do banco de dados). Após o cadastro do projeto, o usuário pode exportar um arquivo com os dados preenchidos ou gerar os códigos do projeto cadastrado.

4.1 MODELO ENTIDADE-RELACIONAMENTO

A Figura 1 apresenta a modelagem entidade-relacionamento do GCER.

A tabela “usuario” mantém os usuários que utilizarão o sistema, possuindo apenas dois atributos: “login” e “senha”. Um usuário cadastro pode definir vários projetos na aplicação.

Cada projeto cadastrado na aplicação pode possuir diversas entidades, as quais poderão ser compostas por vários atributos. Os atributos definidos devem ter a coluna “idTipo” preenchida (que referencia a um registro da tabela “tipo”). A coluna “tamanho” da tabela “atributo” mantém a quantidade máxima de caracteres a ser suportada pelo atributo, quando este for do tipo “texto”.

As tabelas “projeto”, “entidade” e “atributo” possuem as colunas “id” (do tipo *big int*, que mantém um inteiro gerado pelo sistema o qual identifica de forma única o registro), “nome” (do tipo *varchar2* de 50 caracteres de preenchimento obrigatório), “descricao” (do tipo *varchar2* de 200 caracteres, onde o usuário pode definir um comentário ou explicação sobre o item), “dataCadastro” (data gerada pelo sistema quando o registro é inserido), “dataUltimaAlteracao” (data gerada pelo sistema quando o registro é alterado). Além da chave estrangeira para identificar o registro ao qual o item pertence, sendo as colunas: “idUsuarioAutor” na tabela “projeto”, a “idProjeto” na tabela “entidade” e a “idEntidade” na tabela “Atributos”.

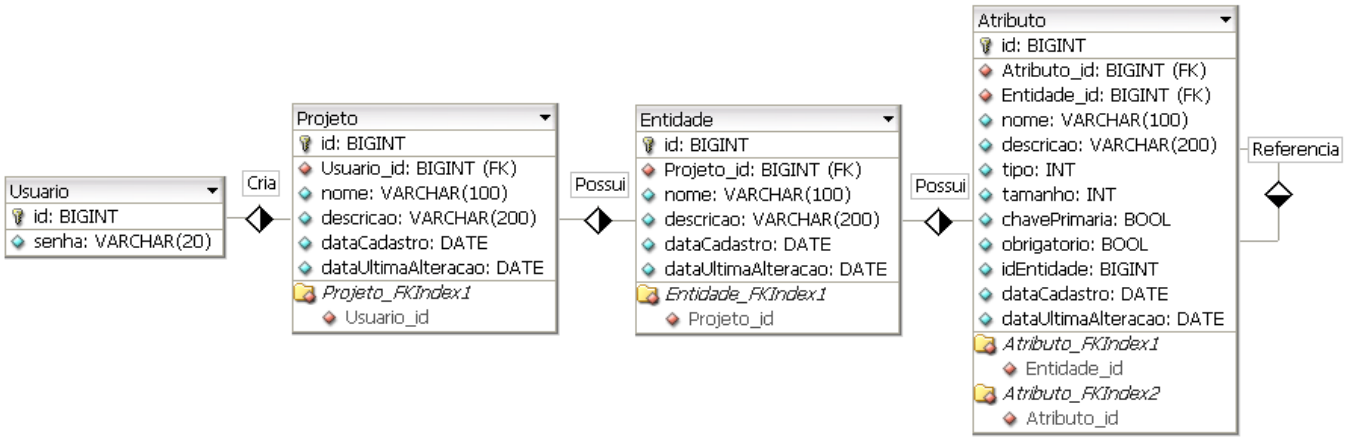

Figura 1 - Modelo Entidade-Relacionamento do Projeto
Fonte - Próprio autor

O sistema também possui uma tabela “tipo”, que mantém os nomes dos tipos dos atributos por linguagem, a biblioteca que deve ser importada para a sua utilização e o tipo da coluna referente ao atributo por banco de dados.

4.2 Modelo de Classes

Foi utilizado o padrão de projeto DAO para o desenvolvimento da aplicação. Segundo Sardagna e Vahldick (2008), o DAO (*Data Access Object*) é utilizado para separar as classes relacionadas ao negócio da aplicação das classes destinadas à persistência dos objetos. Dessa forma, as demais funções do *software* independem, por exemplo, se os objetos do sistema serão armazenados em um arquivo ou em um banco de dados relacional.

Os atributos das classes de entidade “Usuario”, “Projeto”, “Entidade”, “Atributo” e “Tipo”, definidas no modelo de classes, são semelhantes às colunas tabelas “usuario”, “projeto”, “entidade” e “atributo” respectivamente. As diferenças existentes entre as duas modelagens estão apenas nos tipos dos atributos (voltados para os tipos da linguagem Java na modelagem de classes e para os tipos do Oracle no modelo entidade-relacionamento) e para os relacionamentos entre entidades (em que no modelo entidade-relacionamento as tabelas do lado “muitos” da relação possui uma *foreign key* para a chave estrangeira da coluna referenciada, já no modelo de classes há um objeto do item referenciado na classe do lado “muitos” e um *ArrayList* dos objetos do lado “um”).

Para todas as classes de entidade definidas foram criadas uma classe de controle (cujo nome termina com “Control”) e uma classe para a implementação dos métodos para a persistência dos objetos (cujo nome termina com “DAO”). As classes de controle possuem um objeto da classe entidade a que ela se refere e um objeto da sua classe DAO, além dos métodos para salvar, alterar, deletar, buscar e listar os objetos persistidos. Os tratamentos de dados e validações antes da persistência são implementados nas classes de Controle.

Os métodos das classes de controle não possuem os códigos para persistência e acesso aos objetos. Nesses métodos, há apenas a chamada dos métodos da classe DAO da entidade, onde estão implementadas todas as funções de armazenamento e busca no meio persistente (que no caso são o banco de dados Oracle 11g, os arquivos com os códigos gerados e os arquivos exportados).

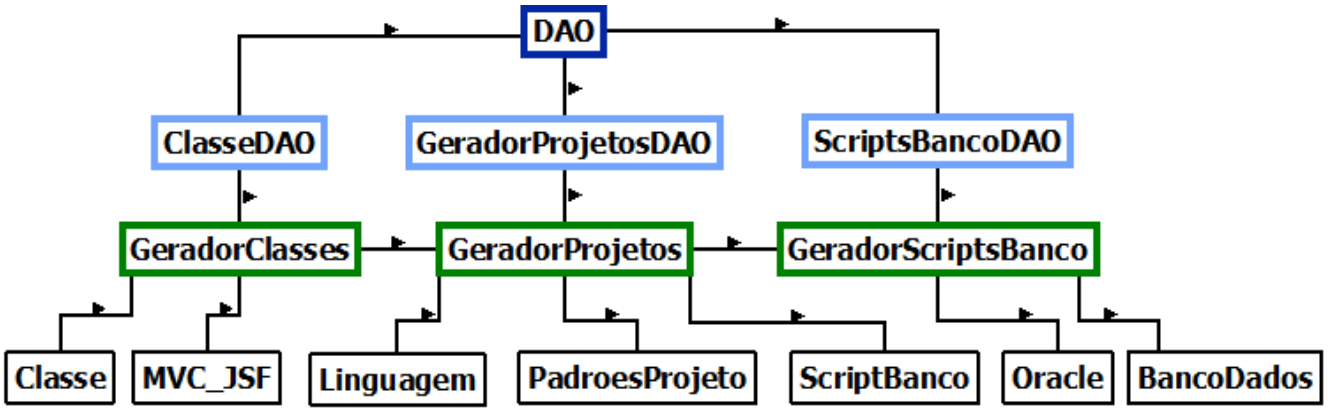

Figura 2 - Modelagem de Classes do Projeto
Fonte - Próprio autor

Todas as classes de persistência referentes a uma entidade herdam da classe DAO, que possui as configurações e métodos para conexão no banco de dados.

A Figura 2 apresenta o modelo de classes para a geração de códigos. Há três níveis de classes abaixo da classe DAO. O primeiro nível é responsável por manter as classes com os comandos para persistência das classes, pastas e *scripts* de banco de dados. O segundo nível de classes possui os métodos que traduzem os dados cadastrados no projeto para os textos a serem escritos nos arquivos. O terceiro nível representa as classes com os dados de entrada para o gerador de códigos. As classes "Classe" e "BancoDados" abstraem os dados do projeto cadastrados pelo usuário. As demais classes "MVC_JSF", "Linguagem", "PadroesProjeto" e "Oracle" possuem os *templates* que fazem os arquivos serem gerados conforme padrões de projeto, linguagens de programação, banco de dados e regras de ferramentas Web (como o JSF) pré-definidas.

Com essa arquitetura de classes, as entidades cadastradas pelo usuário ficam independentes dos estilos de escrita dos códigos. Possibilitando que a aplicação adeque todas as suas funções; reutilize as estruturas das linguagens de programação, banco de dados e padrões de projeto; e seja fácil de adicionar novos componentes.

5 Funcionamento da Aplicação

A aplicação abordada pelo presente artigo possui, como uma de suas principais funções, a possibilidade de um usuário cadastrado definir um modelo entidade-relacionamento para o projeto, ao qual se deseja gerar as implementações das entidades.

Para realizar essa função, o usuário deve:

1. Preencher o nome (obrigatório) e uma descrição (opcional) da entidade e clicar em "Salvar Entidade";
2. Clicar no menu da esquerda "Atributos";
3. Clicar no botão "Novo Atributo" (apresentado na tela da Figura 3);
4. Preencher o nome (obrigatório), selecionar um tipo (obrigatório), preencher uma descrição (opcional), marcar se o atributo será uma chave primária, marcar se o atributo será obrigatório e clicar em "Salvar atributo";
5. Repetir os itens anteriores para adicionar as entidades que o usuário julgar necessárias.

Ao executar a geração de código de um projeto cadastrado, a aplicação primeiramente garante que o usuário autor possua uma pasta, que armazenará as implementações do projeto durante o processamento e escrita dos arquivos dos códigos.

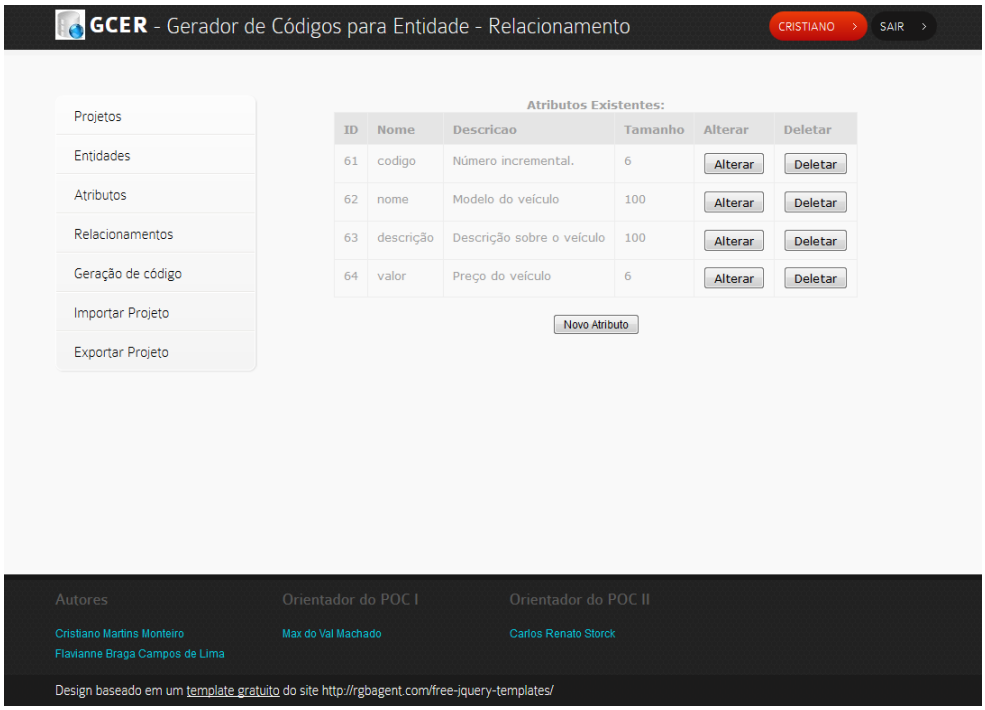

Figura 3 - Tela de Listagem e Manutenção dos Atributos
Fonte - Próprio autor

O sistema valida se já existe no diretório raiz da aplicação uma pasta com o nome do *login* do usuário. Caso não a encontre, essa pasta é criada para armazenar as últimas gerações de código de cada projeto. Cada projeto gerado estará em uma pasta (com o mesmo do projeto) dentro do diretório do usuário.

São geradas então duas subpastas, uma chamada “sql”, que possui os arquivos “create.sql”, “select.sql”, “update.sql” e “delete.sql” com *scripts* para um banco de dados selecionado, e um diretório chamado “src/br/com”, contendo as classes geradas, conforme um determinado padrão de projeto.

Na pasta “sql”, o arquivo “create.sql” mantém os *scripts* para a construção das tabelas no banco de dados. Já os arquivos “select.sql”, “update.sql” e “delete.sql” possuem os comandos para selecionar, alterar e excluir os registros das entidades cadastradas, respectivamente. O banco de dados utilizado como *default* pela aplicação é o Oracle 11g.

As subpastas e arquivos do diretório “src/br/com” dependem do padrão de projeto selecionado. O sistema utiliza como *default* os padrões de projeto DAO e MVC para desenvolvimento de *software* na linguagem Java, utilizando o framework JSF na versão 2.2.

No modelo MVC, aplicado no JSF, as classes de modelo e controle são criadas dentro do diretório “src/br/com” em subpastas com o nome “model” e “control” respectivamente. As classes relacionadas à persistência de dados estão incluídas na subpasta “dao”, localizada dentro da pasta “model”. Os arquivos de tipo “.xhtml”, referentes à visualização estão na pasta “WebContent”, criada dentro da pasta do projeto.

A seção 6 mostra os resultados obtidos pelo sistema desenvolvido. São apresentados exemplos de artefatos de código gerados para uma classe do modelo, seu formulário Web para cadastro e o diretório do projeto com as todas as pastas criadas e organizadas pelo GCER.

6 RESULTADOS

A Figura 4 mostra a árvore de pastas geradas pela aplicação utilizando os padrões de projeto MVC com DAO para um projeto de sistema para venda de veículos. As Figuras 5 e 6 apresentam, respectivamente, um trecho da classe modelo "Veiculo" e o *script* do formulário "xhtml" para a entidade "Veiculo". Os códigos gerados foram compilados ou interpretados adequadamente.

As classes de modelo geradas incluem um comentário informando a data de criação e o usuário autor, declaração do pacote em que a classe está contida, importação das bibliotecas utilizadas, implementação dos atributos privados e seus métodos modificadores e de acesso e definição do construtor sem parâmetros e do construtor com todos os atributos definidos.

Os arquivos com os formulários Web criados utilizam os recursos do *framework* JSF. Nesses arquivos são declaradas, inicialmente, as bibliotecas necessárias e definidas as *tags* para cada atributo modelado. Todos os campos de formulário criados possuem as propriedades "id" (cujo valor é o nome do atributo), "value" (referente ao atributo do *bean* definido para a entidade), e " required="true" " (caso o atributo seja de preenchimento obrigatório).

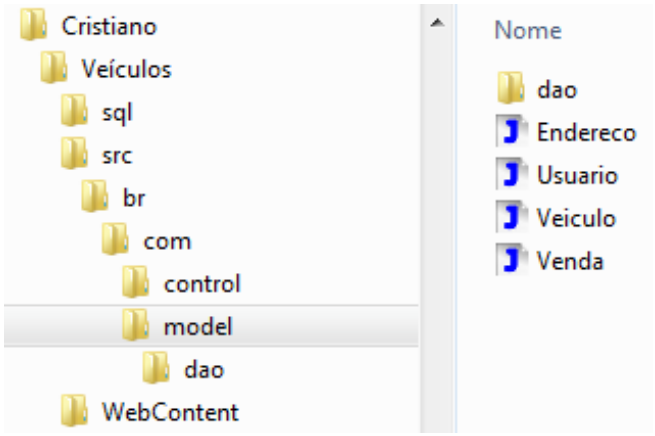

Figura 4 - Estrutura das Pastas Criadas
Fonte - Próprio autor

```
/*
    Código gerado automaticamente pelo GCER versão 1.0
        Autor: Cristiano
        Data de criação: 09/10/2013
*/

package br.com.model;

import java.util.Date;

public class Veiculo {

    private int codigo;
    private String nome;
    private String descricao;
    private Date dataCriacao;
    private double valor;
    private int codigoVendedor;

    public Veiculo() {
    }

    public Veiculo(int codigo, String nome, String descricao, Da
        this.codigo = codigo;
        this.nome = nome;
        this.descricao = descricao;
        this.dataCriacao = dataCriacao;
        this.valor = valor;
        this.codigoVendedor = codigoVendedor;
    }

    public int getCodigo() {
        return codigo;
    }

    public void setCodigo(int codigo) {
        this.codigo = codigo;
    }
```

Figura 5 - Trecho da Classe Modelo da Entidade "Veiculo"
Fonte - Próprio autor

7 CONCLUSÕES

Este trabalho apresentou o desenvolvimento do GCER, um sistema Web para a geração automática de códigos de acordo com o modelo de dados, linguagem de programação e padrão de projeto definidos pelo usuário.

Os arquivos gerados foram compilados sem erros, comprovando a viabilidade da ferramenta proposta para a agilizar as tarefas rotineiras na implementação de sistemas Web.

A arquitetura de classes proposta pôde acomodar facilmente as diferentes fontes de *template* provindas de linguagem de programação, banco de dados, ferramenta de desenvolvimento Web, e padrão de projeto.

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:ui="http://java.sun.com/jsf/facelets">

    <h:form>
        <h:inputText id="codigo" value="#{veiculo.codigo}" required="true"></h:inputText>
        <h:inputText id="nome" value="#{veiculo.nome}" required="true"></h:inputText>
        <h:inputText id="descricao" value="#{veiculo.descricao}"></h:inputText>
        <h:inputText id="dataCriacao" value="#{veiculo.dataCriacao}" required="true"></h:inputText>
        <h:inputText id="valor" value="#{veiculo.valor}" required="true"></h:inputText>
        <h:inputText id="codigoVendedor" value="#{veiculo.codigoVendedor}" required="true"></h:inputText>
    </h:form>

</html>
```

Figura 6 - Formulário Web para a entidade “Veiculo”
Fonte - Próprio autor

O foco em criar um gerador de códigos com suporte à escolha do padrão de projeto distingue o GCER dos trabalhos relacionados encontrados na literatura.

8 Trabalhos Futuros

Como projetos futuros, sugere-se:

1. Utilizar *threads* para a definição e escrita dos arquivos;
2. Permitir a geração de código com suporte à herança e polimorfismo entre entidades;
3. Substituir o uso de formulários no cadastro do modelo de dados dos projetos por campos únicos de texto com processamento de linguagem natural.

9 Agradecimentos

Os autores agradecem ao professor Max do Val Machado, autor do escopo inicial do projeto, e ao orientador Carlos Renato Storck, pela constante participação e incentivo no desenvolvimento do sistema.

Referências

ADAMATTI, M. P.. **FUMIGANT: Gerador de Código Java a Partir da Base de Dados**. 2006. 77 f. Monografia – Faculdade Cenecista Nossa Senhora dos Anjos, Gravataí, 2006. Disponível em: <http://docplayer.com.br/3772196-Fumigant-gerador-de-codigo-java-a-partir-de-base-de-dados.html>. Acesso em: 20 mai. 2016.

ARAÚJO, M. A. P. **Modelagem de Dados – Teoria e Prática**. Saber Digital: Revista Eletrônica do CESVA, Valença, v. 1, n. 1, p. 33-69, mar./ago. 2008. Disponível em: <http://www.lbd.dcc.ufmg.br:8080/colecoes/sbes/2001/020.pdf>. Acesso em: 12 abr. 2013.

ARIATI, A.. **Sistema web para Armazenamento e Recuperação de Artefatos de Software**. 2011. 111 f. Monografia – Universidade Tecnológica Federal do Paraná, Pato Branco, 2011. Disponível em: <http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/615/1/MD_COADS_2011_1_05.pdf>. Acesso em: 20 mai. 2016.

CASTRO, L. L. **Procedimentos de Modelagem e uma Ferramenta de Geração Automática de Código**. Monografia. Universidade Federal de Lavras, Minas Gerais, 2010. Disponível em: <http://www.bcc.ufla.br/monografias/2010/LUCAS_DE_LUCA_CASTRO.pdf>. Acesso em: 02 mai. 2013.

COELHO, L. F.. **Gerador de código HTML baseado em dicionário de dados utilizando banco de dados**. 2006. 52 f. Monografia (Ciência da Computação) – Universidade Regional de Blumenau, Centro de Ciências Exatas e Naturais, Blumenau, 2006. Disponível em <http://campeche.inf. furb.br/tccs/ 2006-I/2006-1luisfernandocoelhovf.pdf>. Acesso em 10 abr. 2013.

FRANCA, L. P. A. **Um Processo para a Construção de Geradores Automáticos**. Tese de Doutorado. Universidade Pontifícia Católica, Rio de Janeiro, 2000. Disponível em:<ftp://ftp.inf.puc-rio.br/pub/docs/theses/00_PhD_franca.pdf>. Acesso em: 02 mai. 2013.

FRANCO, R. S. T.. **Estudo Comparativo Entre Frameworks Java para Desenvolvimento de Aplicações Web: JSF 2.0, Grails e Spring Web MVC**. Monografia. Universidade Tecnológica Federal do Paraná, Curitiba, 2011. Disponível em: <http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/492/1/CT_JAVA_VI_2010_16.PDF>. Acesso em: 06 jun. 2013.

GLOBALCODE. **Projeto OpenSource: SuperCRUD**. São Paulo: Globalcode, 2009. Disponível em <http://www.globalcode.com.br/noticias/EntrevistaSuperCrud>. Acesso em: 20 mai. 2016.

MOREIRA, E.; MRACK, P.. **Sistemas Dinâmicos Baseados em Metamodelos**. In II WORKSHOP DE COMPUTACÃO E GESTÃO DA INFORMACÃO (WCOMPI2003), 2, 2003, Santa Cruz do Sul. Disponível em: <http://3layer.com.br/confluence/download/attachments/3571742/sistemasDinamicosEmTempoDeExecucao.pdf>. Acesso em: 10 abr. 2013.

OLIVEIRA, A.M.; MÄHLMANN, L.G. **Ferramenta Case (Gerador de Código .Net)**. Universidade Luterana do Brasil, Canoas, 2010. Disponível em: <http://www.ulbra.inf.br/joomla/images/documentos/TCCs/2010_2/tcc2-si-anderson_michereff_oliveira.pdf>. Acesso em: 20 mai. 2016.

ROSÁRIO, J.; GRANDE, M. S. C.; PAZIN, A.. **Gerador de artefatos para aplicações web**. São Paulo, v. 2, n. 3, 1-2, jun 2011. Disponível em: <http://www. salesianolins.br/universitaria/artigos/no3/artigo9.pdf>. Acesso em 10 de abr. 2013.

SARDAGNA, M.; VAHLDICK, A.. **Aplicação do Padrão Data Access Objecto (DAO) em Projetos Desenvolvidos em Delphi**. 2008. Disponível em: <http://www.inf.ufsc.br/erbd2008/ artigos/2.pdf>. Acesso em 01 de mai. 2013.

TAKAI, O. K.; ITALIANO, I. C., FERREIRA, J. E. **Introdução a Banco de Dados.** 2005. Disponível em: <http://www.ime.usp.br/~jef/apostila.pdf>. Acesso em: 12 abr. 2013.