Gen-JPA: Uma ferramenta dirigida por modelos para geração de código Java/JPA

Rhaylson Silva do Nascimento[1], **Fernando Castelo Branco Gonçalves Santana**[1]

[1]Instituto Federal de Educação, Ciência e Tecnologia do Piauí (IFPI) – Teresina, Piauí

{rhaylson.silva,fernandosantana}@ifpi.edu.br

Abstract. *Modeling is an engineering technique used in the specification and visualization of the various items that compose a software. The UML class diagram presents a static view of the abstractions that form the domain of the problem. The diagrams can be transcribed directly into programming languages, being the foundation for the development of applications. Many tools implements the automatic transformation of models to the code, however, inconsistencies are observed when analyzing the result of this process. The application presented in this work implements a specific engineering, based on free tecnhologies, aiming at the reduction of this type of situation.*

Resumo. *A modelagem é uma técnica da engenharia usada na especificação e visualização dos diversos itens que compõem um sistema de informação. O diagrama de classes da UML apresenta uma visão estática das abstrações que formam o domínio do problema. Tais modelos podem ser transcritos diretamente para linguagens de programação, sendo o alicerce para o desenvolvimento das aplicações. Muitas ferramentas implementam a transformação automática de diagramas ao código, entretanto, observam-se algumas inconsistências quando se analisa o resultado desse processo. A aplicação apresentada nesse trabalho implementa uma transformação específica, baseada em tecnologias livres, visando a diminuição desse tipo de situação.*

1. Introdução

A Linguagem Unificada de Modelagem (UML) (do inglês *Unified Modeling Language*) é uma linguagem de modelagem visual que estabelece vocabulário e regras específicas para construção de modelos que apresentem a estrutura de determinado projeto seguindo o paradigma orientado a objetos [Booch et al. 2006]. Uma de suas principais funções é orientar a construção do *software*, descrevendo de maneira clara os componentes estruturais e comportamentais que serão implementados durante a fase de desenvolvimento.

A geração de código Java a partir de diagramas da UML, além de ser tema de estudo no meio acadêmico, é prática recorrente no mercado. Hoje, muitas ferramentas, dentre elas as três maiores da indústria de software: *Astah*, *Enterprise Architect (EA)* e *Rational Software Architect (RSA)* fornecem suporte a esse tipo de funcionalidade [Magalhaes 2011]. Entretanto, observam-se inconsistências no código resultante dessas transformações. Causada principalmente pela falta de regras específicas de mapeamento para itens mais complexos da UML, uma inconsistência ocorre quando determinado conceito do modelo deixa de ser transcrito para o código. Nesse caso, por conta da perda

de informações, o código gerado apresenta divergência com o projeto da aplicação e com isso algumas restrições do sistema deixam de existir.

Segundo [Magalhaes 2011] e [Gessenharter 2008], as inconsistências mais comuns ocorrem durante a transformação de diagramas de classes, estando relacionadas aos conceitos de agregação, composição e navegabilidade das associações. Observa-se que durante a transformação, agregações e composições são mapeadas para o código como associações simples, enquanto a navegabilidade acaba sendo ignorada por algumas aplicações. Mecanismos de transformação mais detalhados só estão disponíveis em versões pagas desses *softwares*, o que dificulta sua implantação em ambientes de desenvolvimento de pequeno porte, já que uma licença anual pode ultrapassar os dois mil dólares (*Astah Professional* $ 2.500,00 e RSA da IBM $ 2.200,00).

A situação descrita anteriormente pode ser amenizada através de duas ações. Inicialmente, é necessário o uso de padrões e tecnologias que diminuam os custos agregados na implementação de ferramentas de transformação. De preferência, dando-se prioridade àquelas que sejam gratuitas. Outro item a ser observado diz respeito à qualidade do processo de transformação modelo-código disponibilizado pela ferramenta, devendo ser mais amplo, abarcando os diversos itens do diagrama de classes. Todavia, cabe ressaltar que mesmo com um mapeamento mais detalhado, conceitos mais complexos como o de agregação e composição do modelo de classes ainda não possuem uma representação direta na linguagem Java [Gessenharter 2008][Szlenk 2006], tornando-se necessário o uso de tecnologias auxiliares, cuja notação forneça abstrações capazes de mapear a semântica desses elementos para o código.

Levando em consideração o que foi exposto, esse trabalho tem como objetivo apresentar a ferramenta de transformação *Gen-JPA*, cuja implementação visa diminuir a ocorrência de inconsistências modelo-código, com base no paradigma objeto-relacional. A aplicação, implementada utilizando tecnologias gratuitas, é capaz de manter as restrições de projeto, através de uma transformação baseada na linguagem Java em conjunto com anotações e propriedades da *Java Persistence API (JPA)*. Nesse cenário, as classes do modelo passam a representar entidades do *software*, isto é, tabelas de banco de dados relacionais. O processo de engenharia a frente fornecido pela *Gen-JPA* é capaz de automatizar o desenvolvimento das entidades de um sistema Java, podendo ser utilizado nos mais diversos tipos de aplicações.

2. Referencial Teórico

As chamadas metodologias dirigidas por modelos aplicam o conceito de engenharia a frente com o objetivo de automatizar a construção do software e assim diminuir os custos de produção e seu tempo de entrega. A modelagem nesse tipo de abordagem é considerada atividade chave, visto que os diagramas, antes utilizados como meros itens da documentação, passam a ser cidadãos de primeira classe usados como alicerce para a construção do sistema [OMG 2019].

Hoje, a *Model Driven Architecture (MDA)* surge como uma boa alternativa, fornecendo tecnologias e padrões para construção de novas ferramentas de transformação modelo-código, estando totalmente interconectada à definição da UML. No contexto desse trabalho, três tecnologias livres da MDA serviram como base para o desenvolvimento da aplicação: *Meta Object Facility (MOF)*, *XML Metadata Interchange (XMI)*,

Model to Text Transformation Language (MTL).

A MOF fornece uma especificação completa, na forma de metamodelos dos diagramas da UML, facilitando a referência das diversas propriedades de componentes do diagrama de classes. Já o XMI traz uma forma de representar modelos em formato estruturado utilizando XML, o que facilita a interopelabilidade entre as diversas ferramentas de modelagem do mercado. Por fim, a MTL permite a extração dos dados estruturados no formato XMI, realizando a conversão desses para uma saída em texto.

3. Metodologia

Antes da implementação da ferramentas, buscaram-se estudos que descrevessem quais os problemas apresentados por ferramentas de transformação no que tange a geração de código Java a partir de diagramas de classes da UML. O trabalho mais detalhado foi o desenvolvido por [Magalhaes 2011], nele demonstram-se os pontos falhos das três principais ferramentas da indústria durante a transformação UML-Java. O trabalho descreve a transformação dos diversos itens do diagrama de classes para o código Java, bem como o retorno desse para diagramas da UML, ou seja, a base da transformação da *Gen-JPA*. Constatando-se que conceitos mais complexos do diagrama de classes, como agregações, composições e navegabilidades não vinham sendo respeitados durante a geração do código, acarretando no surgimento de inconsistências entre modelo e resultado final.

Para verificar se o problema persistia, analisou-se, entre os meses de Janeiro e Março de 2019 as distribuições gratuitas das ferramentas descritas anteriormente, através de diagramas de classes com conceitos complexos. Constatando-se que a transformação de itens como agregação e composições continua comprometida, mas que a navegabilidade passou a ser respeitada durante a transformação. O mesmo procedimento foi realizado em ferramentas gratuitas [Parada et al. 2011] e [Cgernert 2019] e obteve-se resultado semelhante. Por conta dessas constatações, o mapeamento de agregações e composições, bem como da navegabilidade foram pontos focais durante a implementação da *Gen-JPA*.

4. *Gen-JPA*

O processo de engenharia a frente implementado na *Gen-JPA* é realizado em três etapas: extração das informações do modelo, validação dos dados e geração do código fonte. Cada etapa é executada por um módulo específico da aplicação que utiliza uma ou mais tecnologias definidas na MDA.

Todos os módulos encontram-se integrados dentro de um mesmo programa Java, executado em ambiente *Desktop*, na forma de um formulário *SWING*. Na leitura do diagrama, optou-se pelo uso de modelos estruturados no formato (XMI) [OMG 2019]. Tais modelos devem pertencer preferencialmente ao pacote *model* da aplicação, levando-se em consideração sua natureza persistente. As subseções seguintes descrevem as atividades envolvidas em cada etapa do processo de engenharia a frente implementado.

4.1. Extração dos Dados

O modulo de extração é responsável por percorrer o modelo de entrada, recuperando as informações dos itens de interesse do diagrama (classes, atributos, associações e

heranças), estruturando esses dados em um formato que facilite sua recuperação pelos módulos de validação e construção . Esse procedimento é realizado através de transformações do tipo modelo-texto, onde instâncias de elementos da UML são convertidos em *tags* dentro de um arquivo XML. Durante a extração dos dados, algumas propriedades como nomes de classes e atributos passam por processamento adicional com o objetivo de padronizá-los às recomendações da linguagem Java.

4.2. Validação do Modelo

O módulo de validação da *Gen-JPA* implementa regras que verificam a integridade dos blocos do modelo, identificando aqueles cuja notação apresenta alguma incompatibilidade com o Java ou que ferem construções da UML, como por exemplo, composições cuja cardinalidade seja diferente de um para muitos. Nessa etapa, também são verificados os tipos e nomes dos atributos e classes, bem como a ocorrências de heranças múltiplas. Apenas diagramas de classes bem-formados e em consonância com a linguagem Java são aprovados pelo módulo de verificação, evitando que o código gerado contenha erros que venham a impedir a sua compilação.

4.3. Geração do Código

A geração de códio na *Gen-JPA* é realizada por classes de controle responsáveis por mapear a sintaxe da linguagem Java e algumas anotações da *Java Persistence API*. O processo ocorre em duas fases: mapeamento objeto-relacional e geração de arquivos fonte. Na primeira, os blocos recebem as propriedades da JPA na forma de anotações, já na segunda as classes são agrupadas e passam pela transformação final. Atualmente, a aplicação é capaz de gerar a definição de classes, atributos e métodos *Gets* e *Sets*, bem como o mapeamento de atributos do tipo temporal.

No código fonte, todas as classes do modelo recebem a anotação @*Entity* e um atributo id em conjunto com a anotação @*Id*, indicando sua natureza persistente. O mapeamento da navegabilidade é mantido, respeitando a definição das associações. A figura 1 apresenta as duas primeiras etapas de transformação para um diagrama com associação simples.

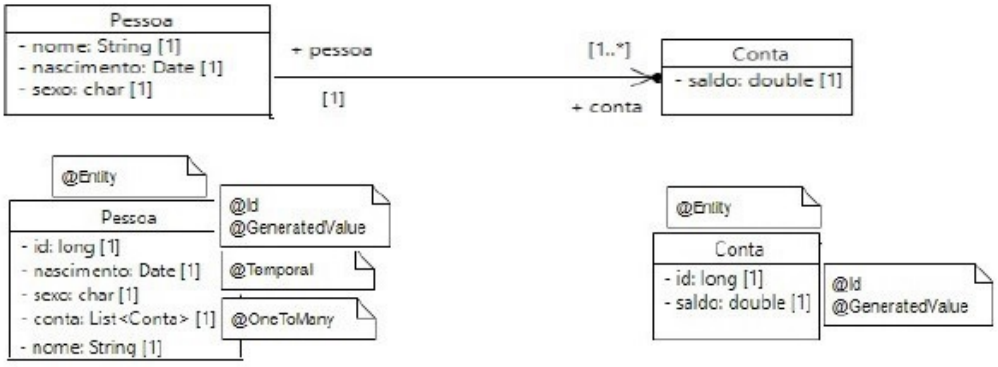

Figura 1: Etapas 1 e 2 da Transformação

Agregações e composições são mapeadas através da propriedade @*Cascade*. Sendo que, as agregações recebem a propriedade *fetch=FetchType.EAGER*, indicando que

a classe necessita ter seus dados complementados pela propriedade após consulta. Enquanto, composições recebem a propriedade *cascade=CascadeType.ALL* indicando que a criação e destruição do atributo é realizado pela entidade que o contém. Tal mapeamento mantém a definição semântica das duas propriedades como consta na definição da UML. A multiplicidade das associações é convertida em atributos mapeados pelas anotações *@OneToOne*, *@OneToMany*, *@ManyToOne* e *@ManyToMany* a depender das especificações do diagrama de entrada. Ao final do processo, cada classe do modelo gera uma classe Java com as devidas importações e métodos de acesso. A figura 2 mostra o código resultante do diagrama anterior com os *import*, *Gets* e *Sets* suprimidos.

```java
@Entity
public class Pessoa {

    public Pessoa(){}

    @Id
    @GeneratedValue
    private Long id;
    @Temporal(TemporalType.TIMESTAMP)
    private Date nascimento;
    private Character sexo;
    private String nome;
    @OneToMany
    private List<Conta> conta;
```

```java
@Entity
public class Conta {

    public Conta(){}

    @Id
    @GeneratedValue
    private Long id;
    private Double saldo;
```

Figura 2: Código Gerado pela Aplicação

As tecnologias empregadas no desenvolvimento da aplicação são gratuitas e disponibilizadas pela *Object Management Group (OMG)* dentro da especificação da *Model Driven Architecture*, já definida anteriormente. O código da aplicação encontra-se disponível para download na plataforma *GitHub* [do Nascimento 2019] sem custos adicionais.

5. Conclusão

A *Gen-JPA* gera código Java/JPA, automatizando a implementação das entidades de uma aplicação. A transformação modelo-código proposta, além de mapear os blocos básicos do diagrama de classes, abrange conceitos mais ricos do modelo (agregação, composiço e navegabilidade), diminuindo a ocorrência de inconsistências. A ferramenta encontra-se disponível de forma gratuita na plataforma *GitHub* [do Nascimento 2019] e todas as tecnologias empregadas no seu desenvolvimento fazem parte da MDA. O código gerado pelo processo pode ser utilizado nos mais diversos ambientes, tais quais *Web*, *Mobile* e *Desktop*.

Referências

Booch, G., Rumbaugh, J., and Jacobson, I. (2006). *UML: guia do usuário*. Elsevier Brasil.

Cgernert (2019). Database development with hiberobjects. `https://techieexchange.wordpress.com/2008/02/07/database-development-with-hiberobjects/`. [Online; acessado 24-março-2019].

do Nascimento, R. S. (2019). Gen-jpa. https://github.com/Rhaylson/Gen-JPA. [Online; acessado 11-outubro-2019].

Gessenharter, D. (2008). Mapping the uml2 semantics of associations to a java code generation model. *Model Driven Engineering Languages and Systems*, pages 813–827.

Magalhaes, L. P. A. (2011). Um estudo sobre a engenharia de ida e volta entre uml e java. Master's thesis, UNIVERSIDADE FEDERAL DE MINAS GERAIS.

OMG (2019). Mda - the architecture of choice for a changing world. http://www.omg.org/mda/. [Online; acessado 15-julho-2019].

Parada, A. G., Siegert, E., and de Brisolara, L. B. (2011). Gencode: A tool for generation of java code from uml class models. In *Proc. 26th South Symposium on Microelectronics (SIM 2011)*, pages 173–176.

Szlenk, M. (2006). Formal semantics and reasoning about uml class diagram. In *Dependability of Computer Systems, 2006. DepCos-RELCOMEX'06. International Conference on*, pages 51–59. IEEE.