GENCODE: UMA FERRAMENTA PARA GERAÇÃO DE CÓDIGO JAVA A PARTIR DE DIAGRAMAS DE SEQUÊNCIA E DE CLASSES

PARADA, Abilio[1]; SIEGERT, Eliane[1]; BRISOLARA, Lisane[1]

[1]Universidade Federal de Pelotas, Centro de Desenvolvimento Tecnológico
{agparada,esiegert,lisane}@inf.ufpel.edu.br

1 INTRODUÇÃO

O desenvolvimento de software embarcado exige que os projetistas considerem aspectos da plataforma do hardware, e trabalhem com restrições rígidas de memória, desempenho e consumo de energia. Além disso, é necessário que o produto seja desenvolvido rapidamente para que se torne competitivo. Ao mesmo tempo, a complexidade do software embarcado nos produtos aumenta constantemente seguindo as exigências por maior sofisticação.

Uma abordagem para lidar com projeto complexos é o uso de modelos [Selic, 2003]. Nesta linha, a linguagem UML tem sido utilizada, pois suporta um alto nível de abstração e permite representar diferentes visões de um sistema. A Engenharia Orientada a Modelos (MDE) [SELIC, 2006] pode proporcionar, ao mesmo tempo, abstração e automação [TERRIER, 2007], [BRISOLARA, 2009], dois aspectos buscados por projetistas de software embarcado. Nesta abordagem, modelos são refinados e transformados até que seja possível a obtenção automática de uma solução mais concreta. No entanto, para o uso efetivo de MDE, é necessário ferramentas que capturem modelos, transformam esses modelos, e geram código.

Este artigo apresenta uma ferramenta para suportar o uso do paradigma MDE no processo de software embarcado, a qual é capaz de gerar código Java a partir de diagramas UML de sequência e de classes.

O artigo está organizado da seguinte forma. Na Seção 2 é apresentada a metodologia usada para geração de código. A Seção 3 apresenta um estudo de caso e discute os resultados que podem ser obtidos com o uso desta ferramenta. As conclusões e trabalhos futuros são apresentados na Seção 4.

2 METODOLOGIA

Esta seção apresenta detalhes sobre a ferramenta GenCode, bem como a metodologia adotada para a geração de código. A ferramenta GenCode (GENerator CODE), implementada na linguagem Java, suporta o uso efetivo do paradigma MDE e é capaz de gerar código Java a partir de modelos UML, incluindo parte estrutural e comportamental. A versão atual suporta diagramas de classes e de sequência, sendo que suas funcionalidades básicas são divididas em dois passos, a captura do modelo (representado em XMI), e a geração do código Java. A geração de código é uma conversão dos elementos capturados do modelo para texto ou código. Esta técnica é usual, e também é usada por [USMAN, 2009].

A partir das informações do diagrama de classes, a ferramenta gera código Java estrutural, que consiste basicamente na declaração das classes, seus métodos e atributos. Os atributos definidos no modelo são declarados de acordo com seus tipos e são geradas assinaturas para os métodos. Além de métodos declarados no modelo, também são gerados o método construtor e métodos Get e

Set para acessar atributos. Outros atributos também são gerados para representar relacionamentos entre as classes (associação, agregação ou composição). Além disso, a ferramenta gera declaração de pacotes, de super classe (se houver herança) e de realização quando houver relacionamento entre classes e interfaces.

A geração do método construtor inclui invocação do construtor da superclasse, se necessário, inicialização dos atributos da classe, por passagem de parâmetros na assinatura do método construtor. Quando uma classe possui atributo com cardinalidade "vários", um Array do objeto do atributo é inicializado, considerando também atributos provenientes de relacionamentos. Na geração da assinatura dos métodos, são consideradas as definições de parâmetros que podem ser do tipo *in, out, inout* ou *return* e define-se a visibilidade, e tipos para todos os parâmetros. Quando há uma hierarquia de herança, os métodos abstratos herdados são gerados como métodos concretos no código das subclasses concretas.

Diagramas de sequência representam o comportamento de um sistema através de troca de mensagens entre objetos, suportando também a representação de laços e condicionais. Estes laços e condicionais serão convertidos para comandos como *for*, *while*, *switch*, *if, etc*. Quando um diagrama de sequência representa o comportamento de um método, as sequências das invocações dos métodos são usadas para gerar o código correspondente a este método. Quando um fragmento do tipo *ref* é encontrado, um outro diagrama de sequência deve ser capturado para gerar o código referente. Em nossa abordagem, a geração do código comportamental inicia com o diagrama de sequência correspondente ao método Principal e segue o fluxo de referências a outros diagramas para gerar o código global. O código comportamental gerado não é completo, uma vez que apenas diagramas de sequência são utilizados, e estes tratam basicamente de invocações de métodos, não permitindo representar uma atribuição de variáveis, por exemplo.

3 RESULTADOS E DISCUSSÃO

Esta seção apresenta um estudo de caso que será usado para demonstrar as funcionalidades da ferramenta proposta. No estudo de caso, uma aplicação de Agenda foi modelada na ferramenta Papyrus [Papyrus 2011] usando diagramas de classes (estrutural) e de sequência (comportamental). Este modelo é, então, usado como entrada na ferramenta GenCode.

O diagrama de classes ilustrado na Fig 1 representa a visão estrutural da aplicação, a qual é composta de quatro classes concretas (Agenda, InterfaceDoUsuario, Email, e Fone) e uma classe abstrata nomeada Contato. Esta classe abstrata possuiu duas subclasses, Email e Fone. Neste modelo, é possivel observar a associação entre a classe Agenda e as classes InterfaceDoUsuario e Contato. A classe InterfaceDoUsuario é a principal classe do sistema, a qual possui o método estático *main*, e também outros métodos como adicionar, excluir, editar, e procurar, que representam as funcionalidades da Agenda.

O comportamento do sistema foi descrito usando vários diagramas de sequência (sd), mas por limitação de páginas, apenas o comportamento parcial será apresentado neste artigo. Nas figuras 2-a e 2-b, o comportamento dos métodos *main* e *procurar* da classe são ilustrados. No diagrama Principal (Fig. 2-a), que descreve o método *main*, o fragmento *alt* descreve as opções de funcionamento da Agenda. Quando a opção é igual a "1", a funcionalidade Adicionar é escolhida, para "2" Excluir, se for "3" Editar, e "4" para Procurar. O comportamento do método Procurar está detalhado no diagrama da Fig. 2-b, no qual o objeto *interfaceDoUsuario* invoca

o método Procurar da classe Agenda, passando como parametro o nome do contato a ser buscado. O retorno deste método é atribuido a variavel contato , que será posteriormente passada como parâmetro para o método *visualizarContato*.

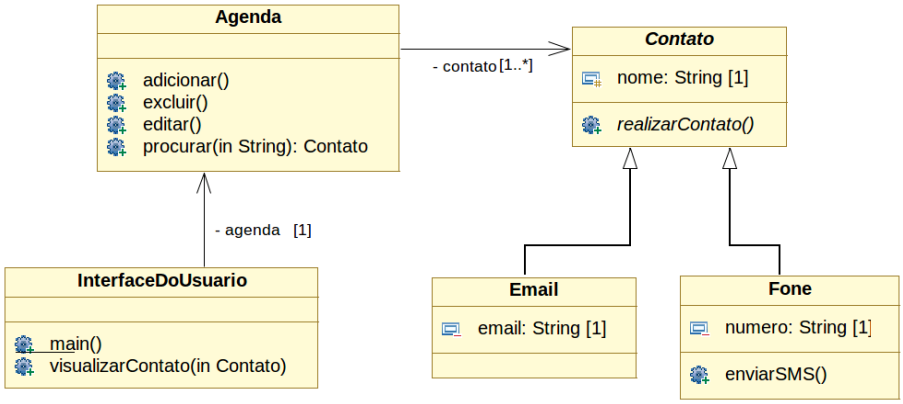

Figura 1. Diagrama de Classes – Visão estrutural da Agenda

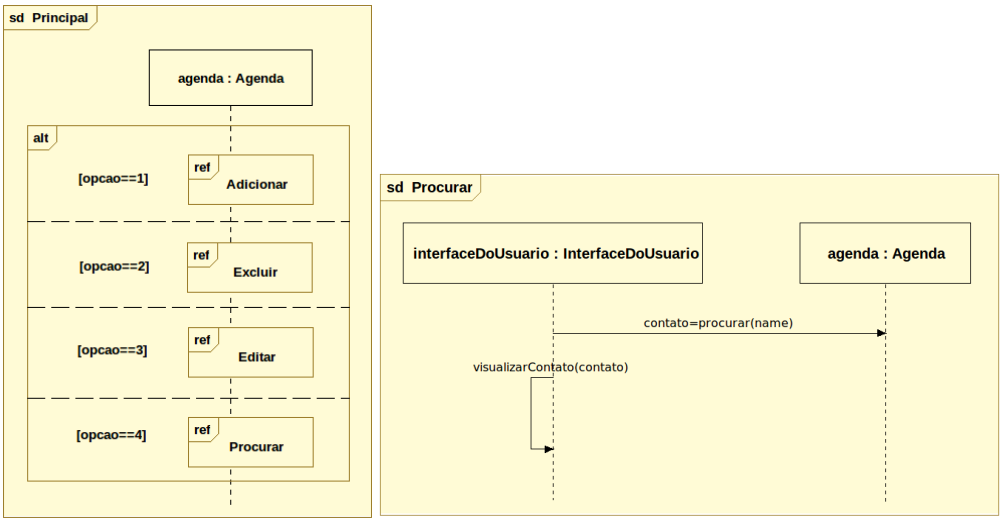

Figura 2. Diag. de sequência – Visão comportamental dos métodos Principal (a) e Procurar (b)

Para demonstrar a ferramenta, foi gerado código para o modelo UML da Agenda. A Fig. 3 ilustra o fragmento de código referente à classe InterfaceDoUsuario, iniciado pela declaração da classe (linha 2) e de seus atributos (linha 5). Na assinatura do método construtor (linha 8), nota-se a passagem de argumentos por parâmetro para inicializar o atributo da agenda (linha 9). Métodos Get e Set são encontrados nas linhas 13 e 18, respectivamente. A Fig. 3 também foi usada para ilustrar a geração de código comportamental gerados a partir de diagramas de sequência. Nesta figura, se pode observar o código referente ao método *main,* gerado a partir do diagrama sd Principal (Fig 2-a). Neste código, observa-se a assinatura do método na linha 23 e a seguir, na linha 25, encontra-se o *switch* que descreve o comportamento representado pelo fragmento *alt* do sd Principal. Como código comportamental é gerado a partir de vários diagramas de sequência, o estudo de caso demonstra ainda o código gerado a partir do sd Procurar (Fig. 2-b) e descrito nas linhas 38 à 42. Na linha 40, observa-se a invocação do método *procurar* da agenda e também a atribuição do retorno do método à variável contato, logo após, na linha 41, observa-se a chamada do método *visualizarContato* da classe interfaceDoUsuário.

```
1
2  public class InterfaceDoUsuario{
3
4    /* Atributos*/
5    private Agenda agenda;
6
7    /* Construtor*/
8    public InterfaceDoUsuario( Agenda agenda ){
9      this.agenda = agenda;
10   }
11
12   /* Get*/
13   public Agenda getAgenda(){
14     return this.agenda;
15   }
16
17   /* Set*/
18   public void setAgenda( Agenda agenda ){
19      this.agenda = agenda;
20   }
21
```

```
22    /* Métodos*/
23    public static void main( String args[] ){
24        /* Especificado pelo Diagrama de Sequência Principal*/
25        switch(opcao){
26            case 1:
27                /* Especificado pelo Diagrama de Sequência Adicionar*/
28                break;
29
30            case 2:
31                /* Especificado pelo Diagrama de Sequência Excluir*/
32                break;
33
34            case 3:
35                /* Especificado pelo Diagrama de Sequência Editar*/
36                break;
37
38            case 4:
39                /* Especificado pelo Diagrama de Sequência Procurar*/
40                contato = agenda.procurar(nome);
41                visualizarContato(contato);
42                break;
43
44            default:
45                break;
46        }//endSwitch
47    }
```

Figura 3. Fragmento de código gerado para a classe InterfaceDoUsuario

4 CONCLUSÃO

Este artigo apresenta uma ferramenta para suportar o uso do paradigma MDE no processo de software embarcado, a qual é capaz de gerar código Java a partir de modelos UML, descritos por diagramas de classes e de sequência. A abordagem proposta permite a geração completa da parte estática do código (declarações das classes, seus atributos e métodos) e a geração parcial da parte comportamental do código. A limitação na geração do código comportamental deve-se a abstração do comportamento descrito em diagramas de sequência que se baseia em invocações de métodos. Como trabalhos futuros, pretende-se estender a ferramenta para suportar outros diagramas e viabilizar a geração de código mais completo, bem como tratar restrições específicas de software embarcado.

5 REFERÊNCIAS

BRISOLARA, L.; KREUTZ, M. E.; CARRO, L. UML as front-end language for embedded systems design. In: GOMES, L.; FERNANDES, J. M. (Org.). **Behavioral Modeling for Embedded Systems and Technologies: Applications for Design and Implementation**. Hershey: IGI Global, 2009. Cap. 1, p. 1-23.

OMG. Object Management Group. UML 2. Disponível em: <http://www.omg.org>. Acesso em: Junho, 2011.

PAPYRUS. (2011) Papyrus 1.12. Disponível em: <http://www.papyrusuml.org>. Acesso em Jan., 2011.

SELIC, B. UML 2: A model-driven development tool. Model-Driven Software Development. **IBM Systems Journal**, Riverton, v. 45, n. 3, p. 607-620, 2006.

SELIC, B. Models, software models, and UML. UML for real: Design of embedded realtime systems (pp. 1-16). Boston: **Kluwer Academic Publishers**, 2003.

USMAN, M.; NADEEM, A. Automatic generation of Java code from UML diagrams using UJECTOR. **International Journal of Software Engineering and its applications** (IJSEIA), v. 3, n 2, p. 21-37, 2009.