Ferramenta MVCASE - Estágio Atual: Especificação, Projeto e Construção de Componentes

Antônio Francisco do Prado
Daniel Lucrédio
e-mail: prado@dc.ufscar.br

Resumo

Este artigo apresenta a ferramenta CASE MVCASE, que suporta o desenvolvimento de softwares orientados a objetos, baseados em componentes. Na ferramenta, o Engenheiro de Software modela o sistema, segundo o método *Catalysis*, e gera seu código para uma plataforma distribuída, usando componentes distribuídos com a tecnologia *Enterprise Java Beans* (*EJB*). A ferramenta disponibiliza os componentes em um *browser*, que facilita seu reuso, através da herança ou instanciação de suas classes.
Palavras-chave: Componentes, CASE, Orientação a Objetos

1. Introdução

A reutilização é um princípio essencial na área de Engenharia de Software para garantir a redução de esforços e custos no processo de desenvolvimento de software. Na tecnologia orientada a objetos a reutilização de software pode ser assegurada com a adoção de *patterns*, *frameworks* de domínios específicos, e com componentes de software já existentes e testados.

Este artigo apresenta a ferramenta CASE MVCASE [Sbes2000], que suporta o desenvolvimento de softwares baseados em componentes. Na sua primeira versão, a MVCASE suporta a especificação de requisitos de sistema em alto nível de abstração, na linguagem orientada a objetos UML[Uml97,Uml99]. No estágio atual, a ferramenta suporta a especificação e projeto de componentes de software, baseado em *Catalysis* [Dso98,Cat2001], seguindo seus 3 níveis: Domínio do Problema, Especificação dos Componentes, e Projeto Interno dos Componentes. Nessa nova versão, a ferramenta MVCASE também suporta a utilização da tecnologia *Enterprise Java Beans* [EJB2001], para construção de componentes distribuídos.

2. A Tecnologia *Enterprise JavaBeans* (EJB)

A tecnologia EJB é um modelo de servidor de componentes para Java. Suporta a criação de aplicações *multi-tier*, que requerem serviços de gerenciamento de transações, segurança, conectividade e acesso a banco de dados. Na tecnologia EJB, as regras de negócio são implementadas nos componentes *beans* que são disponibilizados no servidor EJB, ficando independentes de plataforma.

Cada componente *bean* possui duas interfaces de acesso: interface *Remote* e interface *Home*. A interface *Home* serve como ponto inicial de contato para uma aplicação cliente. Esta interface disponibiliza os métodos para localizar, criar e remover instâncias de um componente *bean*. Quando uma aplicação cliente acessa um componente *bean*, o servidor retorna uma referência para o objeto implementado da Interface *Home*.

A interface *Remote* disponibiliza os métodos das regras de negócio do componente *bean*. Ela recebe a chamada da aplicação Cliente, delegando a execução para uma instância do

componente *bean* e retorna uma referência para o objeto que implementa a interface *Remote*. A Figura 1 mostra, por exemplo, uma tela da MVCASE com a criação do componente *bean*, denominado Cliente, usando a tecnologia EJB.

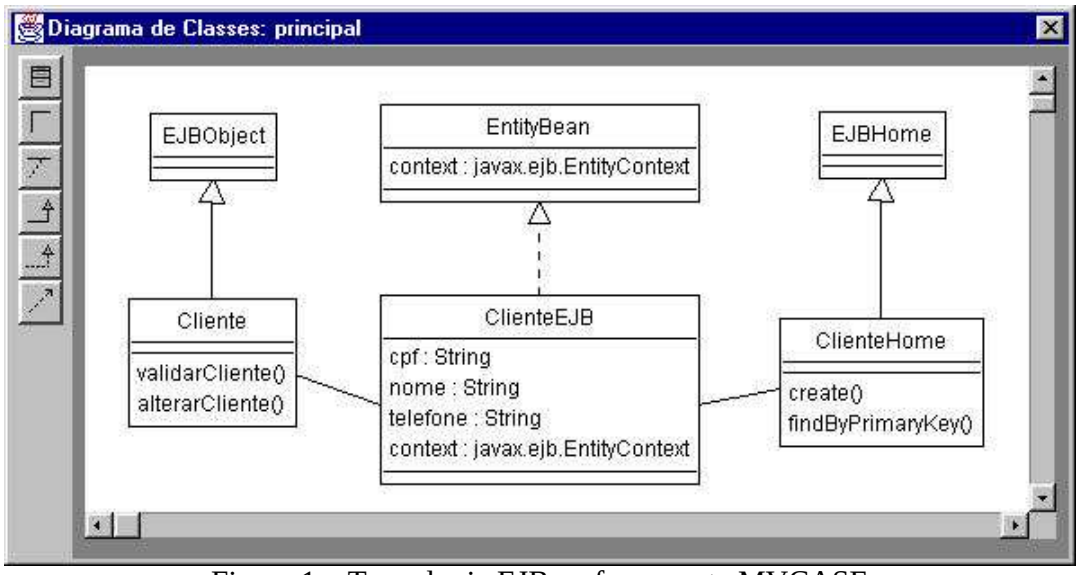

Figura 1 – Tecnologia EJB na ferramenta MVCASE

Na Figura 1, a interface *Cliente* estende a interface *EJBObject* do pacote EJB, e disponibiliza os métodos das regras de negócio do componente, constituindo a interface *Remote* do *bean*. A interface *ClienteHome* estende a interface *EJBHome* do pacote EJB, e é responsável por localizar, criar e remover instâncias do componente, constituindo a interface *Home* do *bean*. A classe *ClienteEJB*, que implementa a interface *EntityBean* do pacote EJB, encapsula todos os atributos e métodos do componente, incluindo aqueles disponibilizados em suas interfaces.

A MVCASE, diferente de ferramentas similares, como por exemplo a *Rational Rose*[Rat2001], oferece a possibilidade da criação automática das interfaces *Home* e *Remote* de um *bean*, a partir da classe contendo as regras de negócio. Dessa forma, obtém-se maior rapidez na construção dos *beans*.

3. MVCASE na Especificação, Projeto e Construção de componentes

A primeira versão da ferramenta MVCASE [Sbes2000], inteiramente construída na linguagem Java, suporta especificações de requisitos de software em alto nível de abstração, na linguagem orientada a objetos UML[Uml97,Uml99]. A partir desta versão, novos recursos foram adicionados na MVCASE. Atualmente, a ferramenta também suporta o desenvolvimento de softwares baseados em componentes segundo o método *Catalysis* [Dso98,Cat2001].

A utilização da MVCASE, com o método *Catalysis*, resulta num processo eficiente para modelagem de sistemas baseados em componentes. Esse processo é dividido em 4 níveis: Domínio do Problema, Especificação dos Componentes, Projeto Interno dos Componentes e Geração de Código Distribuído. Segue-se a apresentação de um estudo de caso, que exemplifica este processo. Trata-se de um sistema de Distribuidora de Produtos.

3.1. Domínio do Problema

No nível **Domínio do Problema** são especificados os requisitos do sistema, preocupando-se com "o que" o sistema deve fazer para solucionar o problema. Identificam-se os tipos de objetos e ações, agrupando-os em diferentes visões por áreas de negócio. Os casos de uso do sistema são identificados e modelados em diagramas de Casos de Uso, conforme mostra a Figura 2. Os relacionamentos dos atores, no caso 'Cliente' e 'Fornecedor', com os casos de uso, são indicados, dispensando, neste nível, detalhes que não são relevantes para o contexto do sistema.

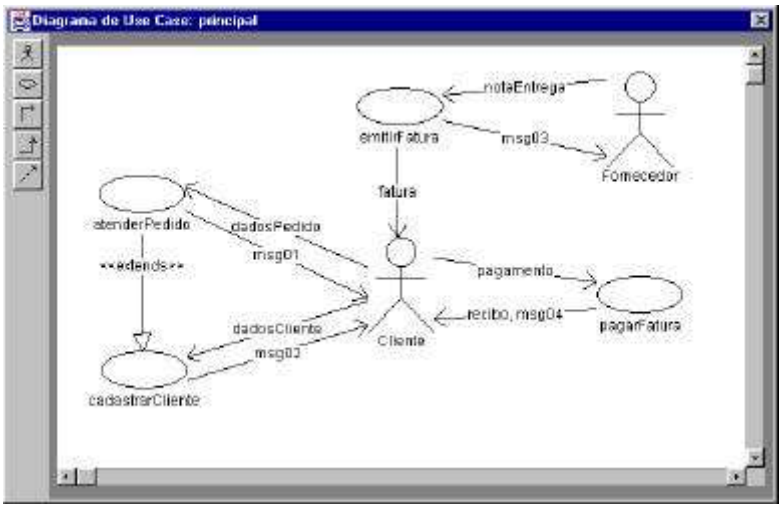

Figura 2 – Modelo de Casos de Uso

Também pode ser utilizado um diagrama de seqüência para detalhar um caso de uso, conforme mostra a Figura 3, para o caso de uso 'atenderPedido'. Neste nível do problema, são indicadas somente as conexões de mensagens relevantes para o contexto do sistema.

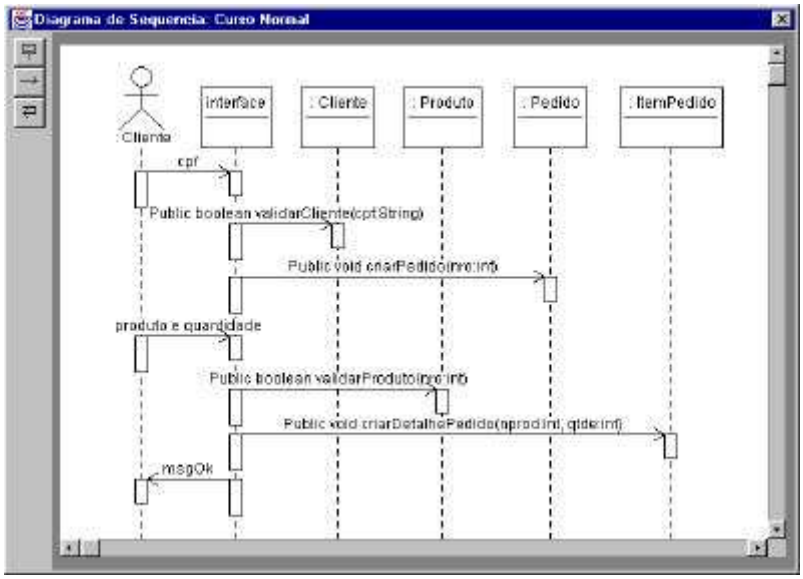

Figura 3 – Diagrama de Seqüência

3.2. Especificação dos Componentes

Nesse nível, refinam-se os modelos do nível **Domínio do Problema**, especificando os componentes do sistema. O Engenheiro de Software preocupa-se com a identificação, comportamento, e responsabilidades dos componentes. Novos modelos, mais detalhados, são obtidos, porém ainda sem se preocupar com a implementação. A Figura 4 mostra um diagrama de classes com a especificação do componente Cliente. No modelo da Figura 4 tem-se que o componente Cliente disponibiliza seus métodos através das interfaces *InterfaceRemotaCliente* e *InterfaceLocalCliente*.

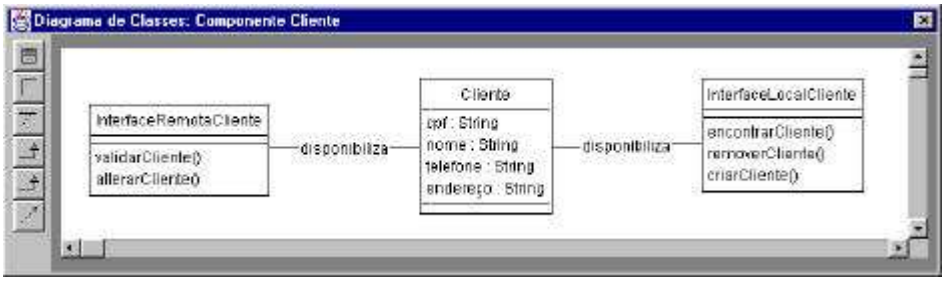

Figura 4 – Especificação do componente 'Cliente'

3.3. Projeto Interno dos Componentes

No terceiro nível de *Catalysis*, o Engenheiro de Software faz o projeto interno dos componentes, dando ênfase às suas implementações e distribuições físicas. Nesse caso, a tecnologia EJB foi utilizada para a construção dos componentes, com características distribuídas e persistência em banco de dados. A Figura 5 mostra o projeto interno do componente Cliente.

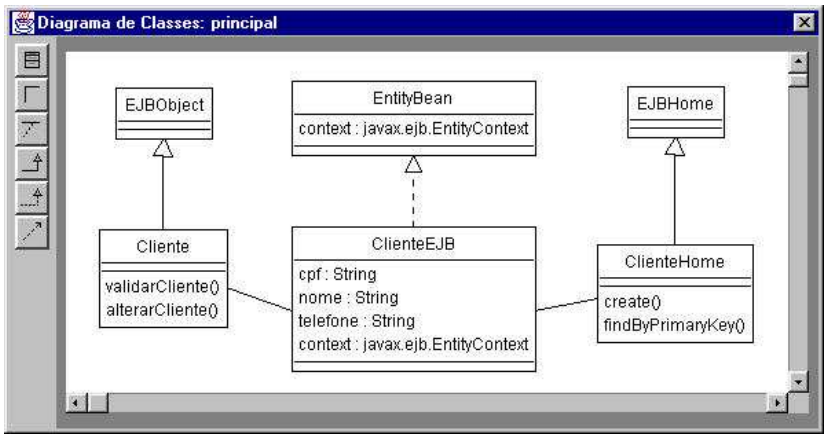

Figura 5 – Projeto interno do componente Cliente

As interfaces do componente especificadas no nível anterior, tiveram seus nomes e métodos modificados para concordar com a tecnologia EJB. No caso *InterfaceRemotaCliente* foi renomeada para *Cliente* e *InterfaceLocalCliente* para *ClienteHome*. O sufixo 'EJB' também foi adicionado na classe do *bean*. As relações de heranças com as classes *EJBObject*, *EntityBean*, e *EJBHome*, do pacote EJB, fazem reuso de componentes do pacote EJB.

Uma outra visão do sistema é obtida com o diagrama de Componentes, que mostra a dependência entre os componentes do sistema. Por exemplo, na Figura 6, o componente *FramePrincipal* utiliza recursos dos componentes *FramePedido*, *FrameProduto*, *FrameCliente*, *FrameFornecedor*.

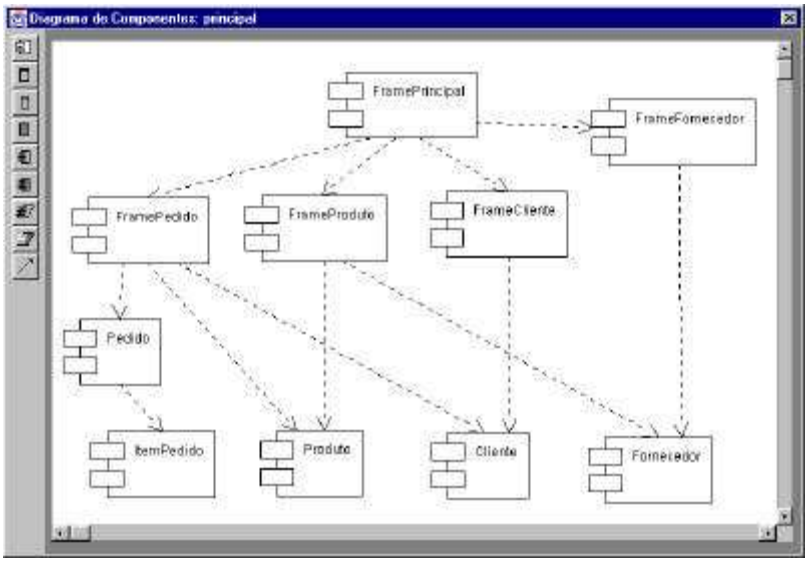

Figura 6 – Diagrama de Componentes

3.4. Geração de código

Após concluída a modelagem do sistema, pode-se gerar o código, na linguagem JAVA, e executá-lo em uma plataforma distribuída Cliente e Servidor, conforme mostra a Figura 7.

```
public class ClienteEJB implements
EntityBean
{
   public String cpf;
   public String nome;
   public String telefone;
...
   public boolean
validarCliente(String cpf)   {
      if(cpf!=null)
      {
         ...
        if(home.encontrarCliente(cpf))
           return false;
        return true;
      }
   }
...
   public String ejbCreate(...) throws
CreateException    {
      this.cpf = p_cpf;
      ...
      return null;
   }
...
```

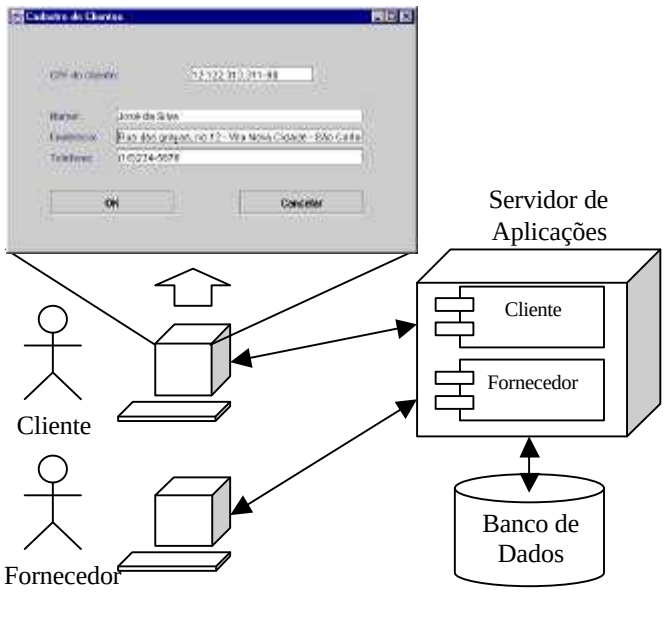

Figura 7 – Código gerado e execução do sistema

À esquerda tem-se o código gerado pela MVCASE, e à direita a sua execução, em um ambiente distribuído. A ferramenta gera automaticamente todo o código dos componentes. O Engenheiro de Software pode verificar se o sistema atende aos requisitos especificados. Caso não atenda, pode reespecificar, reprojetar e reconstruir os componentes, gerando o novo código.

4. Conclusão

O uso da MVCASE no desenvolvimento de softwares baseados em componentes mostrou-se viável e também eficiente. Trabalhando em um alto nível de abstração, o Engenheiro de Software pode se concentrar nos aspectos mais relevantes dos componentes. Além disso, as técnicas para Especificação, Projeto e Construção de componentes de *Catalysis*, facilitam a comunicação entre diferentes Engenheiros de Software, tornando o trabalho em equipe mais produtivo.

A geração automática de código torna o processo de construção de componentes mais rápido, e mantém a consistência entre a implementação e a documentação. A manutenção do sistema fica facilitada, considerando que as alterações nos modelos, em alto nível, refletem automaticamente no código gerado.

5. Referências Bibliográficas

[Dso98] D'Souza, D.; Wills A. *Objecs, Components, and Frameworks with UML: The Catalysis Approach.* Addison-Wesley, 1998.

[Cat2001] Catalysis, Enterprise Components with UML
URL: http://www.catalysis.org, 2001.

[EJB2001] Enterprise Java Beans technology, Sun Microsystems.
URL: http://java.sun.com/products/ejb/index.html, 2001.

[Rat2001] URL: http://www.rational.com/rose, 2001.

[Sbes2000] Prado, A. F., Lucrédio, D; MVCASE: Ferramenta CASE Orientada a Objetos - Sessão de Ferramentas do XIII Simpósio Brasileiro de Engenharia de Software - SBES'2000. João Pessoa–PB, Brasil. 4 – 6 de Outubro, 2000.

[Uml97] Fowler, M.; Scott, K. *UML Distilled – Applying the Standard Object Modeling Language,* Addison-Wesley, 1997.

[Uml99] URL: http://www.rational.com/uml/references, 1999.